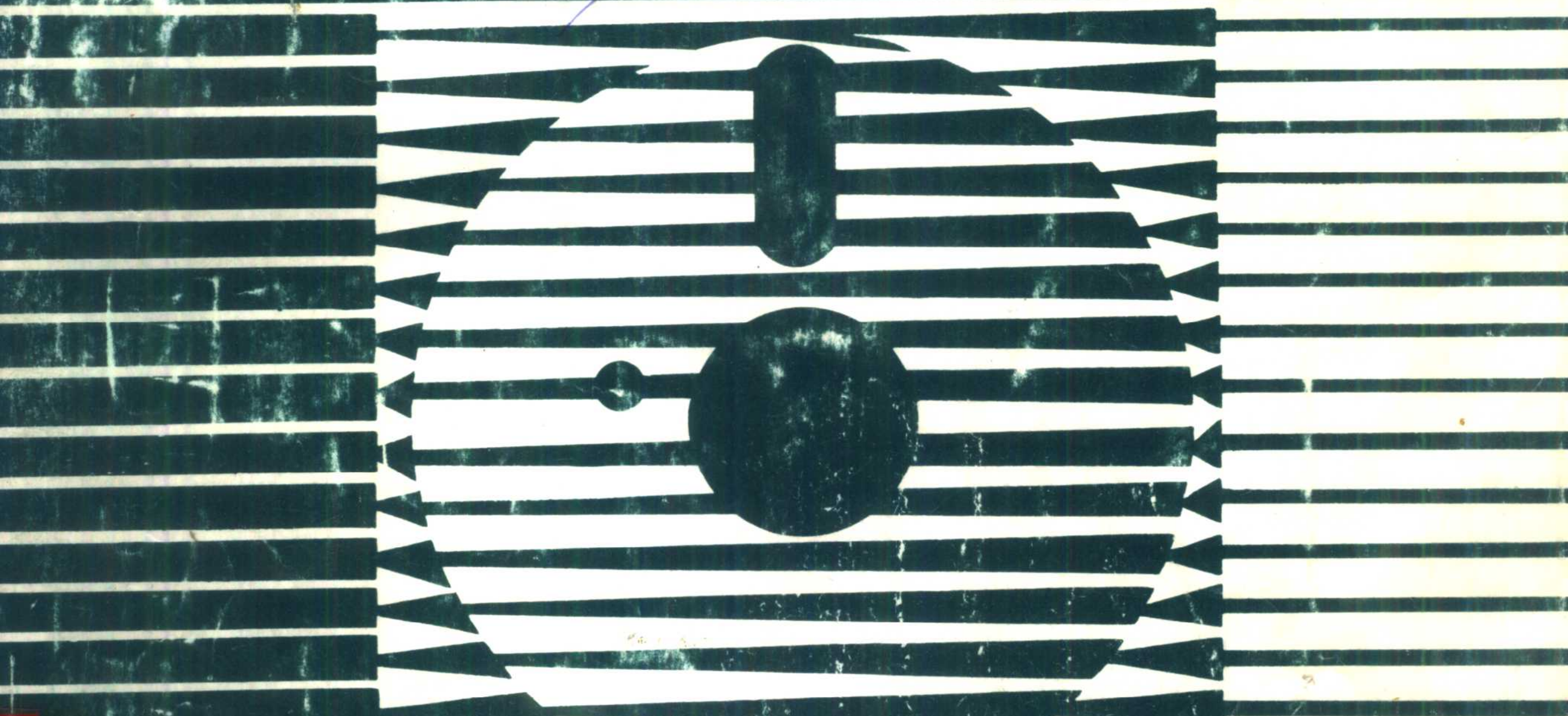
TRUE BASIC
程序设计教程

李世民　潘晓今　杨卫东　刘华红　编著



北京工业大学出版社

True BASIC
程序设计教程

李世民　潘晓今
杨卫东　刘华红　编著

北京工业大学出版社

1990

内 容 简 介

本教程包括 11 章和 7 个附录，系统全面地介绍 True BASIC 及其结构化程序设计。除基本的程序设计内容如输入和输出、判断和循环、数组和矩阵、子程序和数组外，还包括递归技术、大型程序模块化设计、“菜单”驱动、交互式动画、文件处理等高级程序设计的概念与技巧，以及常用的典型算法、汉字应用、程序调试等实用技术。本书在最新推出的 True BASIC 2.03 版本基础上写成，包括了 1.00 版本所没有的许多新功能和特性。

本书结合教学特点，叙述由浅入深，循序渐进;书中包括大量新颖的程序实例和习题，便于学习和思考。本书既可作为大专院校计算机基础教学的教材，也可作为计算机技术人员和爱好者进行 True BASIC 应用程序设计的实用参考书，还可供工程技术人员和管理干部自学使用。

True BASIC 程序设计教程

李世民　潘晓今　杨卫东　刘华红　编著

特约编辑　罗春元　刘宜河　罗献林

北京工业大学出版社出版
新华书店北京发行所发行
中国有色金属工业总公司矿产地质研究院印刷厂印刷

开本 787×1092 1/16　印张 14.6　字数 372 800
1990 年 12 月第 1 版　1990 年 12 月第 1 次印刷
印数: 0001~4500 册
ISBN 7-5639-0119-1/TP·5
定价: 7.00 元

前言

计算机是当代最重要的科学技术成果之一，它对社会的发展起着越来越重要的作用。计算机软件作为知识产品，是现代信息社会的重要物质基础。计算机作为智力工具，它在教育事业中占有特殊的位置。

随着计算机应用与教育的深入，从计算机软件生产和教学中发展起来的软件规范化、程序设计工程化的技术思想，已经为广大计算机工作者和教师普遍接受和积极推崇。Pascal 程序设计语言作为结构化程序设计语言的代表，在计算机专业教育中已经确立了它的重要地位。但是，对于更为广大的非计算机专业的计算机程序设计教学而言，选择一种既与已经广泛应用的程序设计语言相联系，吸取它们的优点，又符合结构化设计理论要求的新的程序设计语言作为学习的对象，当是最恰当的。由美国计算机专家 John G.Kemeny 和 Thomas E.Kurtz 严格按照美国国家标准局制定的 BASIC 标准设计，于 1985 年发表的《True BASIC》，即是这样一种理想的程序设计语言。它为具有极其广泛应用基础的 BASIC 开辟了新纪元。

True BASIC 保持了传统 BASIC 简明灵活的特点，并吸收了 FORTRAN、Pascal 等语言的长处，具有程序模块结构清晰，语法表达自然等优点。不但程序的解释执行与程序的编译结合得很好，而且程序的编辑与连接也十分方便。因此，无论是在工程科学计算、数据处理还是图形绘制方面，都能很好地满足计算机程序设计基础教学，以及科学技术和管理工作的需要。

True BASIC 于 1986 年初在我国一出现，就引起了我们的重视，很快即引入我院的计算机基础教学。1987 年初，为给桂林冶金地质学院地质矿产勘查专业硕士研究生讲授程序设计课程，由李世民编写了《True BASIC 程序设计》简明讲义。之后，经李世民、潘晓今、杨卫东等反复修改、补充，继续在研究生和本科生各专业的“程序设计”教学中试用了 3 年，并先后作了 4 次修订重印，逐步形成了一套以结构化设计方法结合典型算法为特点的《True BASIC 程序设计》教材，内容和程序例题方面体现了 True BASIC 最新版本的性能。本教程即为我们多年来所用教材全面系统整理的结果。

本书内容涉及面较广。除基本的程序设计内容，如：输入、处理、输出、判断和循环、数组和矩阵、子程序和函数过程外，还叙述了一些具有启发性的程序模块化设计、“菜单”驱动、交互式动画、声音音乐处理、文件处理、递归技术等高级程序设计的概念与技巧，以及常用的典型算法、汉字应用、程序调试等实用技术。

本书共分 11 章和 7 个附录。第 1 章由李世民、潘晓今编写，第 2 章由刘华红编写，第 3～8 章由李世民编写，第 9、第 10 章由潘晓今编写，第 11 章及附录 1～3 由杨卫东编写，附录 4～7 及全书的习题由刘华红编写，最后由李世民统修定稿。

OJS138/08

书中带 * 号的章节，例如模块、图画、记录文件、字节文件及一些例题等，有一定的难度，但对高级程序设计有帮助。当教学时数偏紧或对于初学者，可以跳过这些内容，亦可将其作为课程设计的参考材料。

中南工业大学陈松乔教授详细审阅了书稿并提出了宝贵意见，在此谨表诚挚谢意！对支持和组织本书出版的桂林冶金地质学院教务处、学报编辑部、教材科、计算机教研室等部门表示衷心的感谢；对帮助整理书稿的尹晟等老师表示谢意。

由于编著者水平所限，书中难免存在缺点错误，敬请专家和广大读者指正。

编　著　者
于桂林冶金地质学院
1990年7月

目　　录

前　言

第 1 章　电子计算机概述

1.1　计算机的发展和应用 …… (1)

1.2　计算机基本结构及其功能 …… (2)

1.3　计算机软件的概念 …… (4)

1.4　程序设计的概念 …… (6)

习题 1 …… (9)

第 2 章　True BASIC 初步

2.1　True BASIC 的基本语句与程序结构 …… (10)

2.2　True BASIC 的基本操作 …… (14)

习题 2 …… (18)

第 3 章　量与赋值

3.1　数值常量与字符串常量 …… (19)

3.2　简单变量与下标变量 …… (21)

3.3　赋　值 …… (23)

习题 3 …… (26)

第 4 章　函数和表达式

4.1　系统函数 …… (28)

4.2　表达式 …… (36)

习题 4 …… (39)

第 5 章　输入和输出

5.1　INPUT 语句 …… (40)

5.2　PRINT 语句 …… (42)

5.3　READ / DATA 语句与 RESTORE 语句 …… (46)

习题 5 …… (49)

第 6 章　判断结构

6.1　算法及其逻辑框图表示 …… (51)

6.2　IF 型判断结构 …… (52)

6.3　SELECT　CASE 判断结构 …… (56)

6.4　程序举例 …… (58)

习题 6 …… (62)

第 7 章　循环结构

7.1　FOR / NEXT 循环和 EXIT　FOR 语句 …… (63)

7.2　DO / LOOP 循环和 EXIT　DO 语句 …… (67)

7.3　程序举例 …… (70)

习题 7 …… (75)

第 8 章　数组与矩阵

8.1　数组说明与数组遍历 …… (77)

8.2 数组的应用 …… (82)
8.3 矩阵语句 …… (86)
8.4 内部数组和矩阵函数 …… (93)
8.5 程序举例 …… (97)
习题 8 …… (105)

第 9 章 子程序、函数和库

9.1 子程序 …… (107)
9.2 函数 …… (115)
9.3 库 …… (121)
9.4 递归 …… (123)
*9.5 公用变量、共享变量和模块 …… (126)
9.6 程序举例 …… (132)
习题 9 …… (138)

第 10 章 图形

10.1 屏幕模式与窗口设置 …… (140)
10.2 基本绘图语句 …… (142)
10.3 图形着色 …… (147)
10.4 BOX 语句与动画设计 …… (152)
*10.5 图画 …… (154)
*10.6 坐标输入、单键输入和多重窗口 …… (158)
*10.7 声音与音乐 …… (164)
*10.8 程序举例 …… (166)
习题 10 …… (182)

第 11 章 文件

11.1 概述 …… (184)
11.2 文件的公共操作 …… (184)
11.3 正文文件 …… (187)
*11.4 记录文件 …… (193)
*11.5 字节文件 …… (197)
11.6 打印机的使用 …… (198)
11.7 高级编辑 …… (199)
习题 11 …… (201)
附录 1 出错处理 …… (203)
附录 2 程序的设计与调试技术 …… (206)
附录 3 True BASIC 的运行环境 …… (209)
附录 4 命令总结 …… (212)
附录 5 语法补充 …… (216)
附录 6 出错信息 …… (220)
附录 7 ASCII 码表及 GET KEY 键码表 …… (225)

参考文献

第 1 章　电子计算机概述

本章介绍计算机的工作原理、计算机的硬件和软件、计算机系统组成及程序设计概念等内容，为更好地掌握程序设计技术和进一步学习计算机科学打下基础。

1.1　计算机的发展和应用

1.1.1　计算机发展概况

现代科学技术的进步，于1946年诞生了世界上第一台电子计算机--ENIAC，从而使计算工具发生了质的跃变。40多年来、电子计算机技术性能的不断提高，应用范围的不断扩大，极大地推动了社会生产、社会管理、社会服务各个领域中科学技术的迅速发展，使计算机科学成为科技进步的带头学科和先导技术。它正在促使今天的世界由工业化社会向信息化社会转化。

自第一台电子计算机诞生以来的40多年中，由于对计算机结构原理的研究不断发展和微电子技术不断进步，使电子计算机大约每10年就发生一次换代更新：

1946年～1956年——电子管计算机；

1957年～1963年——晶体管计算机；

1964年～1974年——集成电路计算机；

1975年以后——大规模集成电路计算机。

每次换代，计算机的性能都比上一代优越，运算速度成十倍地提高，而体积迅速缩小，成本大幅度下降。目前，计算机正向两极发展：计算机的微型化，促进了计算机应用的普及与社会化；而计算机的巨型化，则促进了尖端科技的研究与发展。计算机的发展水平已经成为一个国家应用技术水平和科学技术发达程度的重要标志。

1981年10月在日本东京召开的第五代计算机国际会议的情况表明，第五代计算机将是超大容量、超高速、具有新的结构体系的智能化计算机。这一领域已经成为世界科技竞争的主要目标之一。

1.1.2　计算机的应用

由于计算机具有高速度运算、大容量存贮、能进行逻辑判断和自动运行等特点，今天电子计算机的应用已经遍及到社会生活的各个领域。

(1) *科学计算*　高速度、高精度的数值计算是计算机在科学、工程、技术方面最基本和经常的应用。数学、物理、天文、地理、气象、水文、土木工程中许多繁杂的运算工作只有电子计算机才能胜任。

(2) *数据处理与信息加工*　对航天技术、地质勘测、银行帐目、资源管理、人口统计、网络通迅、情报资料检索中大批量的数据传递、分析和处理，高速度、大容量计算机可以应付自如。

(3) *自动控制*　卫星、导弹、飞船、自动生产线和复杂的空中及地上交通管理，都需要实时的自动控制。计算机在这个领域中的应用，使其技术质量与效率显著提高。

(4) *计算机辅助设计和辅助教育*　CAD（计算机辅助设计）用于电子、机械、航空、造船、化工、建筑等部门，使得复杂产品的设计过程能自动产生产品的造型，分析产品的性

能，对设计方案作出选择与优化。因而可以提高设计质量，缩短设计周期，使设计标准化。而 CAI（计算机辅助测试）在学习和培训方面的应用，可以有效地提高教育质量和水平。

(5) 人工智能　其研究的领域包括：模式识别、语言识别、定理自动证明、自动程序设计、专家系统和机器人等。目前最有代表性和实用意义的是专家系统和机器人。

另外，办公自动化和计算机艺术造型，也是近年来迅速发展的计算机应用领域。

1.2　计算机基本结构及其功能

现今使用的计算机绝大部分都是冯·诺伊曼式计算机（Von Neumann 是 20 世纪著 0 名的数学家）。这种计算机主要由 3 类功能不同但相互关联的部件构成：存贮器（Memory）；中央处理器（CPU）；输入／输出设备（I／O Units）。

存贮器是保存信息并能方便地存取信息的装置；中央处理器是对信息进行运算和处理的装置；输入／输出设备是实现计算机与外部环境交换信息的装置。

冯·诺伊曼式计算机的基本结构如图 1-1 所示。常见微型计算机系统如图 1-2。

组成计算机的物理器件实体称为计算机的硬件，硬件分为主机和外部设备两大类。主机主要由中央处理器和内部存贮器组成，它是计算机的中心；主机以外的各种设备（如外部存贮器、输入设备、输出设备等）统称为外部设备。

计算机的整体性能是由主机和各外部设备的性能决定的。计算机的基本性能包括运算速度、内存容量、外存容量、输入／输出设备的数量和种类等。

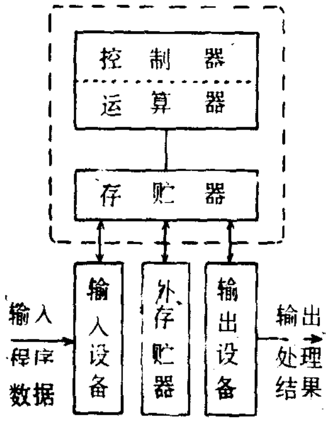


图 1-1　计算机基本结构

1.2.1　存贮器

利用存贮器存取各种信息是现代计算机的重要特征之一。为了既方便又大量地存贮和使用信息，计算机的存贮器分为外部存贮器（简称外存或辅存）和内部存贮器（简称内存或主存）两类。

外部存贮器是独立的外部设备之一，它的主要作用是长期、大量地保存信息。各种备用的程序和数据通常保存在外部存贮器中。计算机工作时，可以根据需要随时将这些程序和数据调入内存使用。目前常用的外部存贮器包括磁盘机、磁带机、光盘机等。

内部存贮器位于主机内部，通常由能够高速存取信息的集成电路芯片组成。它与中央处理器紧密联接在一起。内部存贮器的特点是信息读写的速度极快。中央处理器利用它来存取当前工作过程中频繁使用的信息，以提高计算机的工作效率。内部存贮器又细分为只读存贮器（ROM）和随机存贮器（RAM）两种。计算机启动和运行时必须使用但又不允许修改的信息通常被固化在 ROM 内；而 RAM 中的信息则既可读出又可写入，但是如果关闭计算机，RAM 中的信息亦将随之消失。

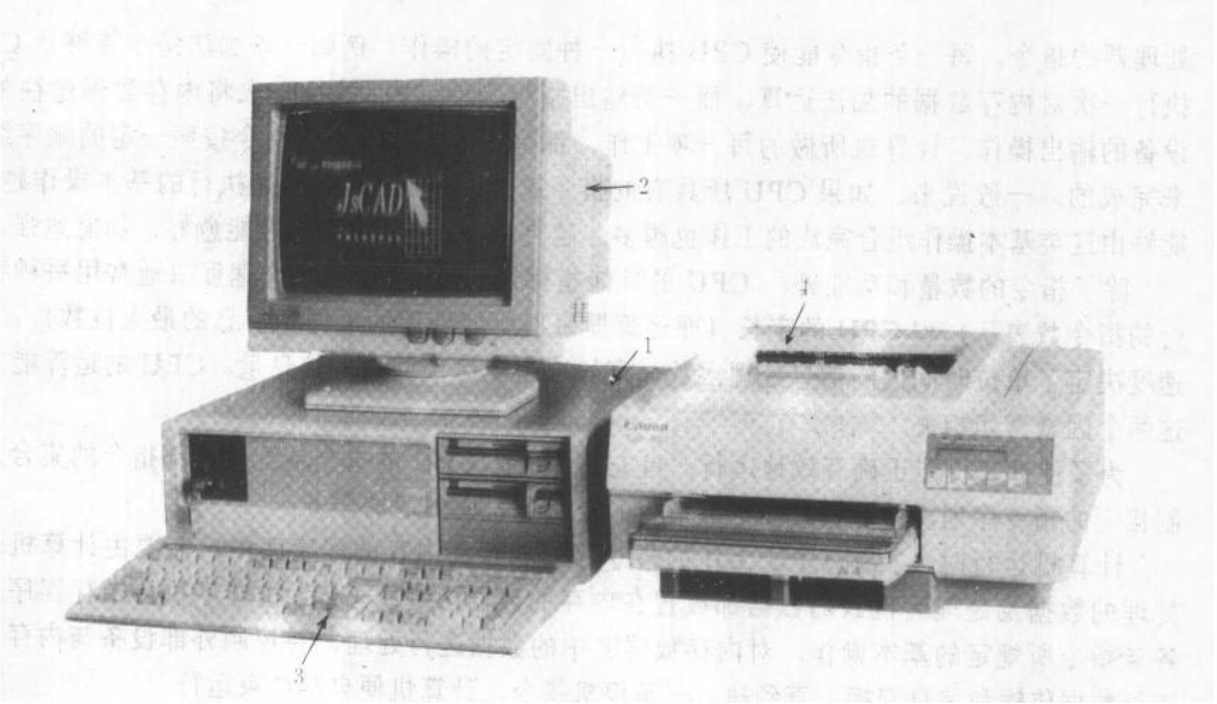


图 1-2　常见微型计算机系统

1-主机；2-显示器；3-键盘；4-打印机

为了能够迅速准确地存取大量信息，存贮器被划分成许多存贮单元。存贮单元具有如下性质：

(1) 每个存贮单元都有一个唯一的编号，通常称为地址。

(2) 每个存贮单元可容纳一个固定位数的二进制信息。二进制信息由二进制数表示。二进制数是由数码“0”和“1”构成的序列，它的每一位（bit）是计算机表示信息的最小单位。每 8 位二进制数为一个字节（byte），一个存贮单元中的信息可由多个字节组成。

(3) 存贮单元中的内容一般既可读出又可写入。对存贮单元进行“读”操作时，仅拷贝其中内容，单元内容保持不变；而对存贮单元进行“写”操作时，是将该单元中原有的内容改写成新的内容，原有信息即被消除。

计算机处理信息，既要求较大的存贮容量，又要求快捷的存取速度。因此存贮容量和存取速度是衡量计算机存贮器性能的两项重要指标。存贮容量常用字节数表示，每 1024 个字节（即 2^{10}）简称为 1KB（即 1024 Byte）。存取速度用读／写 1 个字节信息的时间表示，时间单位为 ms（毫秒，10^{-3} 秒），μs（微秒，10^{-6} 秒）或 ns（毫微秒，10^{-9} 秒）。

1.2.2　中央处理器

中央处理器（简称为 CPU，即 Central Processing Unit）是执行程序、实现信息处理的部件，由运算器（ALU）、控制器（CONT）和若干个寄存器组成。CPU 位于主机内部，它是计算机的核心机构。其中，运算器的主要功能是完成算术或逻辑运算；控制器的作用则是根据程序指令的性质发出各种控制信号，对计算机的工作过程进行控制，以保证计算机的各个部件相互协调地完成对信息的处理；寄存器用于存放当前使用的运算信息和控制信息，它与内存相比具有更快的信息存取速度。

CPU 能够执行若干基本的操作。规定 CPU 执行这些基本操作的二进制信息，称为中央

处理器的指令。每一条指令能使 CPU 执行一种固定的操作。例如一条加法指令能够让 CPU 执行一次对内存数据的加法运算，而一条输出指令则使 CPU 执行一次将内存数据送往外部设备的输出操作。计算机所做的每一项工作，都是由这类基本操作指令按照一定的顺序组合来完成的。一般说来，如果 CPU 所具有的指令越多，说明 CPU 所能执行的基本操作越多，能够由这类基本操作组合完成的工作也越多、越复杂，因而 CPU 的性能愈好、功能愈强。

除了指令的数量和功能外，CPU 的性能指标还包括 CPU 的工作速度（通常用每秒钟执行的指令数表示）和 CPU 的字长（即运算器每次所能处理的二进制信息的最大位数）。工作速度决定了单位时间内的信息处理次数，字长决定了每次处理的信息量，CPU 的运算能力与这两个因素密切相关。

为了使计算机能正确有效地运行，首先需要根据 CPU 的指令系统（全部指令的集合）编制相应的指令序列，使计算机在这组指令的控制下完成预定的任务。

计算机运行时，控制 CPU 工作的指令序列放置在内存的程序区中；需要由计算机进行处理的数据及处理后得到的数据都放置在内存的数据区中。CPU 按顺序执行内存程序区中各条指令所规定的基本操作，对内存数据区中的数据进行处理，并控制外部设备与内存之间进行数据传输和信息交换。直到执行一条停机指令，计算机便自动结束运行。

1.2.3　输入／输出设备

输入设备和输出设备（简称为 I／O 设备，即 Input／Output 设备）是计算机与外部进行信息交换的装置。在 CPU 的控制下，输入设备能够接收来自计算机外部的各种形式的信息（如文字、数字、图形、声音、电讯号等等）。输入设备将这些信息转换成二进制形式，并传输到内存，以便由 CPU 进行处理。同样，输出设备能够在 CPU 的控制下，把计算机内部预先存有或经处理后得到的二进制形式的信息，还原或转换成所需要的信息形式（如文字、数字、图形、声音、电讯号等等），并传输到计算机外部。因此，I／O 设备是实现人及外部环境与计算机相互交流信息的装置。

I／O 设备的配置因计算机不同而异。一台计算机根据需要常常配有多台 I／O 设备。常用的输入设备有：键盘、鼠标器、光笔、数字化仪、图形文字扫描仪、摄像仪、音频输入装置等。常用的输出设备有：屏幕显示器、打印机、X-Y 绘图仪、投影仪、音频输出装置等。另外，计算机之间也能够通过 I／O 设备相互连接，形成一定范围的计算机通讯网络。I／O 设备的种类和性能，是决定该计算机应用范围和功能强弱的重要因素。

1.3　计算机软件的概念

1.3.1　指令与指令系统

计算机的工作过程，就是硬件执行预先设计好的指令序列的过程。一个完整的计算机系统，应当包括计算机硬件和能使硬件有效运行的计算机指令。

不同类型的中央处理器其指令系统是不相同的，但基本包括以下类型的指令：

数据传输指令——将数据从一个部件传输到另一部件；

算术逻辑指令——对数据进行算术或逻辑运算。数据的来源及处理后的存放位置既可是某寄存器，也可以是某存贮单元；

程序转移指令——改变指令的执行顺序；

处理机控制指令——对中央处理器本身进行控制。

在每种类型的指令中，根据特定的操作要求又包含了许多具体指令，因此一台计算机往往包含几十或上百条指令。一台计算机所有指令的集合，称为该计算机的指令系统。

按照一定的规则组成，可完成规定功能的指令序列称为程序。构造该指令序列的过程称为程序设计。程序以及设计、使用和维护该程序的有关文件档案，统称为计算机软件。

1.3.2 软件的种类

计算机软件按用途可分为3类。

(1) *面向计算机管理和操作的软件* 这类软件包括操作系统、分时系统、网络系统等。它们的作用是：当计算机工作时，自动对计算机的硬件设备及有关软件进行有效管理，使其进入正常的工作状态。例如，让计算机的键盘等待输入，以便接收计算机用户输入的信息；让计算机的显示器等待从主机传送过来的信息，以便及时输出到屏幕上；让作为外存使用的磁盘机等待中央处理器的控制信号，以便和内存交换信息等等。一台比较简单的计算机只需要一般的机器管理程序对其进行管理。而一台配置齐全、功能完善的计算机，就需要使用操作系统。

操作系统的主要功能是管理计算机硬件的中央处理机、存储器、外部设备以及外存储器上的软件等。它能最大限度地发挥硬件和软件的作用，使用户能够有效地利用和控制计算机。

对于连接了多个用户终端的计算机，分时系统能够让主机的工作时间合理地分配给各个终端使用。如果建立了计算机网络，那么在网络系统的支持下，便可以实现计算机之间的信息通讯。面向计算机管理和操作的软件与硬件的关系密切，是根据硬件的结构和工作方式来设计的。只有在这些系统软件的支持下，计算机才能够发挥效能，各种其他软件也才能够有效地运行。

(2) *面向计算机维护的软件* 这类软件包括机器故障检查和诊断程序、自动纠错程序、硬件性能测试程序、软件测试程序等。在进行硬件或软件安装、调试、检测、修理时，或者是在计算机运行过程中，它们被用来检查硬件或软件故障，纠正由计算机故障所引起的运算错误。

(3) *面向用户的软件* 这类软件包括计算机语言的处理程序、图形文字等各类信息数据的编辑程序、数据库管理程序、汇集了常用数学处理方法的数学库程序、解决各类应用问题的各种专用或通用的应用程序等。最基本的是各种计算机语言的处理程序。面向用户的软件使计算机的用户能够方便有效地进行程序设计，为用户提供各种运算处理功能，从而有利于解决各种计算机开发应用问题。

计算机语言处理程序包括编译程序和解释程序两种形式。编译程序的作用是将用计算机语言写的源程序全部翻译成计算机能够执行的机器指令代码目标程序。解释程序对源程序的翻译过程是边翻译边执行，不生成目标程序。目标程序运行速度较快，而解释方式有利于人机对话。有了编译程序或解释程序，用户便能够利用易于描述解题过程的计算机语言编制各种应用程序，由编译程序或解释程序将它们转换成计算机能够执行的指令，从而提高程序设计的效率。

各类软件中，有的软件用于保证计算机系统正常工作，发挥和提高计算机系统的运行效率，为计算机系统提供通用的服务功能，这类软件称为系统软件；有的软件则是在计算机应

用领域为解决某些具体的计算或信息处理问题而编制的，这样的软件称为应用软件。不同的计算机可以根据其工作性质和应用需要配置不同的系统软件和应用软件。

1.3.3　软件与硬件的关系

由图 1-3 可见，计算机系统中的硬件和软件以及软件的各部分之间是一种层次结构关系。软件是在硬件基础上对硬件性能的扩充和完善，软件之间则是外层软件需要内层软件为其提供运行的条件。在硬件外每加一层软件，计算机的功能就提高一层。通常把这种功能更强的计算机称为“逻辑机”。

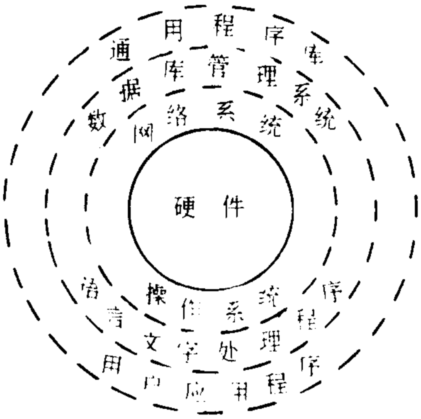


图 1-3　硬件—软件层次结构关系

1.4　程序设计的概念

1.4.1　基本概念

进行程序设计，就是用计算机能识别和理解的符号来描述计算机的运行步骤。程序设计的符号表达规则，称为计算机语言。计算机语言可以大致分为 3 个层次：机器语言，汇编语言，高级语言。在机器语言层次上，直接使用具体的二进制指令编程；在汇编语言层次上，使用易于理解和记忆的指令助记符号编程；在高级语言层次上，则使用接近于自然语言或某种工程语言的表达方式编程。

进行程序设计时，首先需要针对实际问题找出有效的解题方案，即确定解题的“算法”。在计算机科学中，算法往往用自然语言、数学公式、图形符号等易于理解的方式来描述。

算法设计是程序设计的关键。它涉及到合理使用计算机的各种运算操作、合理安排数据和程序在计算机存贮器中的存贮等问题。

为了说明三种不同层次语言的程序设计，这里引入一台简单的模型计算机。

计算机的机器指令的一般格式为：

操作码 （Θ）　　　　操作数 （D）

一条机器指令是一个二进制数，分为操作码和操作数两部份。其中，操作码（Θ）规定该指令的操作性质（即做什么），操作数（D）则给出参加运算的数或参加操作的数之内存地址。

假定这台模型计算机具有表 1-1 所示指令系统。其中，Acc 表示中央处理器中用于放数据和进行算术逻辑运算的寄存器（称为累加器），（Acc）表示该寄存器中的内容；k 表示某个存贮单元的地址，（k）表示该存贮单元中的内容。

1.4.2　计算机语言程序

设要求编写计算分段函数

$$y=\begin{cases} x^2-3, & (x \leqslant 10) \\ x+2, & (x>10) \end{cases}$$

的模型机器语言程序。

在分段函数的计算过程中，x 是输入的变量，2，3，10 是常量，y 是计算结果的输出量。

(1) *机器语言程序*　用具体机器所能识别和执行的机器指令，去描述某个要计算机求解或处理的问题，这样的指令序列就是一个机器语言程序。

为了让计算机能自动按照分段函数的要求，根据输入的 x 求解 y 的数值，需要先将 3 个常数存入内存（例如，将它们分别存入内存的 00001，00010，00011 二进制单元），分配 00100 单元（4 号单元）给变量 x，00101 单元（5 号单元）给变量 y。然后，将按照分段函数的计算要求，用指令编出反映具体计算过程的机器语言程序，把它按指令顺序逐条逐条地存放在从 01010 起的若干个存贮单元之中（每个单元存放一条指令）。内存情况如表 1-2 所示。

(2) *汇编语言程序*　由表 1-2 可见，即使是计算这样一个不太复杂的分段函数，用机器指令所编写的指令代码程序竟如此繁琐。全由“0”和“1”所表示的数据及程序，既容易弄错又不直观。要用这种形式求解一个复杂问题，会存在许多困难。为此，人们创造了汇编语言。汇编语言为计算机的机器语言指令的操作性质安排了助记符号，汇编语言采用这些助记符来写程序；存贮单元可用变量表示，数据可用十进制数表示。计算分段函数的汇编语言程序如下：

```
          INP   x             ;输入 x
          LOA   x             ;取 x 到 Acc
          SUB   10            ;作 x-10
          CJP   L1            ;如果 (x-10) <0 则转标号 L1
          LOA   x             ;取 x 到 Acc
          ADD   2             ;作 x+2
          STO   Y             ;将 x+2 存入 Y
          JMP   L2            ;无条件转向标号 L2
    L1:   LOA   x             ;取 x 到 Acc
          MUL   x             ;将 x 与 x 相乘
          SUB   3             ;从乘积中减去 3
          STO   Y             ;将 x^2-3 存入 Y
    L2:   OUT   Y             ;输出 Y
          HLT                 ;停机
```

显然，汇编语言比机器语言前进了一步：指令符号比较容易记忆，数字、变量表示直观，有利于编制程序，程序的可读性明显地比机器语言的好。

但是，计算机并不能直接执行这种程序。它必须被翻译为机器语言程序后，方能由计算机执行。这个翻译工作就是由一个叫做“汇编语言编译器”的系统软件来完成的。汇编语言所写的源程序，经过“编译器”的翻译，产生一个机器语言程序（又称为目标程序），由计算机执行这个目标程序得到计算结果。

(3) *高级语言程序*　汇编语言虽然比机器语言前进了一大步，但它与特定的机器和指令系统密切相关，程序的通用性较差（当然，这一点将随着硬件的标准化而不断改善）。而且程序设计中需要详细地了解机器的内部特性，才能写出有效的汇编语言程序。为了提高程序设计

表 1-1　一台模型计算机的指令系统

操作码	助记符	操作名	操作内容	指令类型
0001	ADD	加法	Acc ← (Acc) + (K)	算术逻辑指令
0010	SUB	减法	Acc ← (Acc) − (K)	算术逻辑指令
0011	MUL	乘法	Acc ← (Acc) * (K)	算术逻辑指令
0100	DIV	除法	Acc ← (Acc) / (K)	算术逻辑指令
0101	LOA	取数	Acc ← (K)	数据传输指令
0110	STO	存数	K ← (Acc)	数据传输指令
0111	JMP	无条件转移	转向第 K 单元	数据转移指令
1000				
1001				
1010	CJP	条件转移	若 (Acc) <0 则转 K， 否则转下一条指令	程序转移指令
1011	INP	输入	把输入设备上的数送入 K	数据传输指令
1100	OUT	输出	把 K 单元的内容输出外部	数据传输指令
1101	HLT	停机	停止计算结束程序	处理机控制指令

表 1-2　分段函数机器语言程序

地址	RAM 存贮情况	注　　释
00001	00000 00010	常数 2
00010	00000 00011	常数 3
00011	00000 01010	常数 10
00100		存放从外设输入的变量 x 的数值
00101		存放分段函数的函数值 y
00110		
·	·	
·	·	
·	·	
	(Θ)　(D)	
01010	01011 00100	从外设输入 x 的值存入 4 号单元
01011	00101 00100	从 4 号单元取数到 Acc
01100	00010 00100	将 3 号单元中的数与 Acc 中的数相减（即 x−10）
01101	01010 10010	如果（Acc）<0，则转 18 号单元，否则做下一指令
01110	00101 00100	从 4 号单元取数到 Acc（即 Acc← x）
01111	00001 00001	将 1 号单元中的数与 Acc 中的数相加（即 Acc← x+2）
10000	00110 00101	将 Acc 中的数送入 5 号单元（即 Y← x+2）
10001	00111 10110	无条件转向第 22 号单元
10010	00101 00100	从 4 号单元取数送到 Acc（即 Acc← x）
10011	00011 00100	将 4 号单元中的数与 Acc 中的数相乘（即 Acc← x）
10100	00010 00010	将 2 号单元中的数与 Acc 中的数相减（即 Acc← x^2−3）
10101	00110 00101	将 Acc 中的数送入 5 号单元（即 Y← x^2−3）
10110	01100 00101	向外设输出 5 号单元中的数
10111	01101 00000	停机
11000		

的效率，便于程序的修改、调试，人们进一步创造了高级程序设计语言。例如 50 年代的

FORTRAN，60年代的ALGOL-60、BASIC、COBOL，70年代的PASCAL、C，80年代的True BASIC语言等等。

用True BASIC高级程序设计语言来描述上述分段函数的计算过程，就十分简便：

```
INPUT  x                       !输入 x
IF  x<10 then                  !如果 x 小于等于 10 则
    LET  y=x*x-3               !按 y=x*x-3 计算
ELSE                           !否则
    LET  v=x+2                 !按 y=x+2 计算
END IF                         !结束判断处理
PRINT  y                       !输出 Y
END                            !结束程序
```

高级语言是独立于具体机器，便于描述算法的程序设计语言。用高级语言所编写的程序与汇编语言程序一样，机器也是不能直接执行的。但通过一个对应于该高级语言的编译程序或解释程序，能够将其翻译成机器语言的目标程序，再由机器执行。

由上述3种层次的程序设计可以看出，机器语言程序和汇编语言程序是描述计算机内部的具体操作过程的指令程序，因此必须根据具体计算机的指令系统和助记符来进行程序设计。而高级语言程序允许使用类似于自然语言和数学公式的语句序列来描述解题过程，它与具体计算机的指令无关，因此不仅便于描述复杂的算法，而且程序不受具体机器的限制。只要该计算机配有相应高级语言的翻译软件，就可以在指令系统不同的各种计算机上运行高级语言程序。

习 题 1

(1) 计算机硬件由哪几个基本部件组成，各自的主要功能是什么？
(2) 试解释下列概念，并简要说明其相互关系：①软件，②程序，③算法。
(3) 高级语言与机器语言有什么区别，各有什么优缺点？
(4) 解释程序和编译程序有何异同点，它们在计算机系统中起什么作用？
(5) 操作系统在计算机系统中起什么作用？

第 2 章　True BASIC 初步

BASIC 语言自 60 年代创立至今，人们用它解决了大量实际问题，已成为世界上使用得最广泛的高级程序设计语言之一。True BASIC 是 BASIC 的创始人于 1985 年发表的，集其他高级语言之大成，为计算机程序设计提供了又一有效的工具。它具有如下特点：

(1) 结构功能完善，语法表达自然，程序编辑灵活；

(2) 与 BASIC 兼容性良好，在 IBM PC 上可使用全内存；

(3) 系统函数丰富，具有高级图形和动画设计功能；

(4) 解释方式下工作简明方便，编译方式与解释方式并存；

(5) 便于移植，便于与汇编语言程序连接；

(6) 符合美国国家 BASIC 标准；

(7) 支持汉字。

从这一章起将详细讲述 True BASIC 语言及其程序设计。本章主要介绍 True BASIC 程序的整体概貌及 True BASIC 的上机基本操作。

2.1　True BASIC 的基本语句与程序结构

True BASIC 程序由 True BASIC 语句构成，每个 True BASIC 语句规定计算机完成一项特定的工作（ 或称为操作 ）。True BASIC 语句分为简单语句和结构控制语句两种。简单语句使计算机完成一项简明的操作任务，而结构控制语句可以将简单语句组织成各种序列，对程序的运行过程进行控制。

2.1.1　三个简单语句

一般情况下，无论是简单程序或者是复杂程序，都应具备输入数据、运算处理、输出结果这 3 项基本功能。在 True BASIC 语言中，这 3 项基本功能可以运用以下 3 个简单语句来实现。

键盘输入语句：INPUT　Var1，Var2，…，VarN

运算赋值语句：LET　Var1 = expr

屏幕输出语句：PRINT　expr1，expr2，…，exprN

其中，Vari 为变量，expri 为表达式（ 详见第 3、第 4 章 ）。

运用这 3 个最基本的 True BASIC 语句，便能够进行初步的 True BASIC 程序设计。下面是简单的 True BASIC 程序设计的例子。

例 1　从键盘上输入圆的半径值，计算并显示圆的面积。

若用 R 表示圆的半径，S 表示圆的面积，由几何定理有

$$S=\pi\times R\times R$$

根据此数学公式，编制程序如下：

```
INPUT r                          ! 输入圆半径 R 的值
LET s=3.14159*r*r                ! 计算圆面积的值并赋给变量 S
PRINT s                          ! 输出 S 的值
END                              ! 结束程序
```

这个简单程序共有 4 行。第 1 行是键盘输入语句，其作用是要求从键盘上输入一个数据，并将这个数据作为变量 R 的值；第 2 行为运算赋值语句，作用是计算表达式 3.14159 * r * r 的值，然后将计算的结果赋给变量 S；第 3 行为屏幕显示语句，作用是将 S 的值显示在计算机的屏幕上；最后一行是一个专门用于表示程序结束，终止程序运行的语句，称为程序结束语句。显然，程序描述了输入 R，计算 S，输出显示 S，终止运行这样一个完整的计算机工作过程。程序中，按照 True BASIC 语言的语法规则，将常数 π 写成 3.14159，将乘法运算符 × 写成 * 。

程序运行时，屏幕上首先出现一个提示问号“ ？ ”，表示要求从键盘输入数据 R。用户在键盘上将具体数据输入(并按[Return]或 [Enter]键)以后，屏幕上立刻显示出 S 的计算结果。 程序例子中，在语句后面加上了一个“ ! ”号及该语句功能的说明，它们是程序的注释部分，是为了便于初学者理解程序的内容而加的，并不是语句的必要部分，可以不输入。

例 2 求数列 1，2，3，…，N 的和 M:

$$M = \frac{N \times (N+1)}{2}$$

相应的 True BASIC 程序为:

```
INPUT n                          ! 输入 N 的数值
LET m = n * (n+1) / 2            !计算表达式 n * ( n+1 ) / 2 的值，然后赋给变量 M
PRINT m                          ! 输出 M 的值
END                              ! 结束程序
```

这个程序与例 1 的程序类似：首先用键盘输入语句(INPUT 语句)输入 N，其次用运算赋值语句(LET 语句)计算表达式 n * (n+1) / 2 的值(True BASIC 规定程序中除法运算必须用符号“ / ”表示)，并将计算结果赋给 M，然后用屏幕输出语句(PRINT 语句)显示 M 的值。程序结束语句(END)位于最后。

2.1.2 程序格式

通过上面两个例子可以看出， True BASIC 程序的组成是有规则的，可归纳如下。

(1) 程序由语句组成。

(2) 语句具有与操作性质对应的关键字(描述该语句功能的英文单词或英文单词的缩写，如 INPUT、LET、PRINT、END 等)。语句的书写格式以及所使用的符号应当符合 True BASIC 的语法规则。

(3) 语句必须按照执行的先后顺序书写。每个简单语句在程序中占一行。

(4) 程序的最后一个语句必须是 END 语句，并且一个程序只能有一个 END 语句。

进行 True BASIC 程序设计，实际上就是用 True BASIC 语言所提供的各种功能的语句来描述解题过程。如果运算过程比较简单，那么程序可以仅由简单语句组成；如果运算处理过程比较复杂，则需要在程序中使用结构控制语句。

2.1.3 二个结构控制语句

结构控制语句是由以关键字开头和对应的另一关键字结尾的，多个语句行构成的语句组。

True BASIC 由结构控制语句构造 2 种基本的程序结构：判断结构和循环结构。判断结构根据给定的条件，有选择地执行程序中某一部分语句；循环结构则根据需要重复地执行程序中某一部分语句。用这 2 种结构进行合理组合，能使程序具有更灵活的运算处理功能。

(1) 判断结构的形式:

```
IF <c> then
   ......
   ......  } 语句块( 由一个或多个语句组成 )
   ......
END IF
```

其中，<c>表示一个判断条件。

IF(表示判断结构的开始)和 END IF(表示判断结构的结束)构成一个判断结构，也称为 IF / END IF 结构。每个判断结构都有一个判断条件<c>。当程序运行到判断结构时，立即检测 IF 语句中的条件<c>。如果条件成立，运行过程即进入判断结构中的语句块，执行语句块中的各个语句；如果条件不成立，就不执行语句块中的各个语句，而执行 END IF 语句下面的其他程序语句。

例 3 改进例 2 中的程序，使其能够对 N 的输入值进行判断。当条件 $N>0$ 成立时，才根据公式计算并输出 M。

程序:

```
INPUT n
IF n>0 then
   LET m=n*(n+1)/2
   PRINT m
END IF
END
```

改进后的程序之第 2 至第 5 语句使用了 IF / END IF 判断结构，把对 M 进行计算与显示的语句置于此判断结构之内。因此，只有输入的 N 大于 0 的情况才会对 M 进行计算和显示，避免 $N<0$ 时进行计算而出现错误的结果。

(2) 循环结构的形式:

```
DO
   ......
   ......  } 语句块( 由一个或多个语句组成 )
   ......
LOOP
```

DO(表示循环结构的开始)和 LOOP(表示循环结构的结束)构成一个循环结构，也称为 DO / LOOP 循环(或 DO 循环)。DO 至 LOOP 之间的语句块被无限次地反复执行，这种循环过程又称为“死循环”。

例 4 根据不同的半径 R 值计算不同圆的面积 S。

程序:

```
DO
    INPUT r
    LET s=3.14159*r*r
    PRINT s
LOOP
END
```

为了能够不断地输入半径 R 的值并计算出相应的面积 S，程序使用了 DO 循环。事实上，由于 DO 循环语句所构成的重复过程为无限循环，程序一旦开始运行就不断地要求从键盘上输入 R 的值，然后计算并输出 S。程序不可能执行到 LOOP 之后的 END 语句，因此程序将无休止地运行，永不结束。

对于无休止运行的程序，需要人为干预使程序停止运行。强迫程序终止的方法是：同时按下键盘上的[Ctrl]键和[Scroll]键。

为了达到同时按两个键的目的，操作时应当是先按住[Ctrl]键不动，然后按一下[Scroll]键，再同时松开两个键。

一般，循环重复的次数应当是有限的。这就要求循环过程受一定条件的控制，能够按照某种需要退出循环，结束重复过程。退出循环的专用语句是：

```
EXIT  DO
```

执行这个语句，就可从 DO／LOOP 中退出，结束该循环过程，执行 LOOP 后的语句。EXIT DO 应与判断结构配合使用，形成一种“有条件退出循环”的控制机制。

例 5　在例 4 中加入一个判断结构，处理当 $R<0$ 时停止计算圆面积，退出循环。

程序：

```
DO
    INPUT r
    IF r<0 then
      EXIT  DO
    END IF
    LET s=3.14159*r*r
    PRINT s
LOOP
END
```

在程序的 DO／LOOP 循环中，每循环一次就要求输入一个圆半径的值，紧接着就对这个 R 值作判断，当 $R<0$ 时则退出循环，执行 END 语句结束程序运行；对于不满足条件的 R 值就计算并输出对应的圆面积 S，从而使程序能够受控制地结束，避免了死循环。

2.1.4　三种基本的程序结构

由前述例子可以看出，编制 True BASIC 程序使用了 3 种基本的结构形式：顺序结构，判断结构，循环结构。

例 1、例 2 的计算处理过程是由语句的顺序执行来完成的，这就是程序最基本的顺序结构。本质上，一个大程序如果按照它的主要功能将其分为若干顺序相接的程序段，也就形成顺序结构。任何复杂程序都可抽象成按“ 输入—处理—输出 ”模块排列的顺序结构。

判断结构是根据条件有选择地执行不同语句序列的程序段。例 3 和例 5 说明了它的作用。

循环结构是实现重复运算和重复处理的程序段。从例 4 和例 5 可以大致了解它的作用。循环加上“ 有条件退出循环 ”，可以方便地实现有限次的重复过程。

基于上述 3 种基本结构而构造的程序，就是结构化程序。已经证明，任何复杂的程序，都可以用这 3 种结构及其嵌套组合来表示。

2.1.5　程序的书写

书写程序时，控制结构内的各个语句其起始位置应当右移若干个字符，一般右移一个词的位置。采用这种按结构后缩的锯齿形书写格式，是为了使程序的结构层次清晰，增加程序的可读性，便于程序的调试和检查。如果将例 5 写成左边对齐的形式：

```
DO
INPUT r
IF r<0 then
EXIT DO
END IF
LET s=3.14159*r*r
PRINT s
LOOP
END
```

这显然不如例 5 原来的书写形式那样结构清楚和易于阅读。

一般，书写 True BASIC 程序时应当注意：

(1) 一个语句行内的字符个数最多为 32000 个(包括空格)。

(2) 语句关键字、变量数字的中间部位不能有空格。此外，空格可以出现在程序的任何地方。

(3) 字母的大小写可以不加区分(字符串常量除外)。

(4) 为了便于阅读程序，在程序中可以加入注释，注释内容用感叹号“ ！”开头。完整的注释行亦可用 REM 作为注释语句的关键字。

例 6 计算梯形面积 S。

$$S=(\text{上底}+\text{下底})\times\text{高}/2$$

程序：

```
REM This is the program which computes the area of a trapezoid.
! 这是一个计算梯形面积的程序.
DO                                  ! 循环起点
    INPUT ud                        ! 输入上底的值
    INPUT dd                        ! 输入下底的值
    INPUT h                         ! 输入高的值
    IF h<0 then                     ! 如果 h<0 则退出循环
       EXIL DO
    END IF
    LET s=( ud+dd )*h/2             ! 计算梯形面积
    PRINT s                         ! 输出梯形面积 S 的值
LOOP                                ! 返回循环起点
END                                 ! 结束程序
```

作为特例，上面对程序的每个语句都作了注释。一般，仅注释程序的关键部分及重要语句。

另外，与普通 BASIC 比较，结构化的 True BASIC 不需要使用控制转移语句(如 GOTO、GOSUB 语句)，因而也不需要使用行号。但是如果在 True BASIC 程序中使用了行号，则所有语句都必须有行号。行号应为 1～999999 间的整数。行号前不能有空格，行号之后至少要有一个空格。在有行号的情况下，普通 BASIC 的控制转移语句仍然有效。注意：在没有使用 OPTION NOLET 语句声明的情况下，赋值语句的关键字 LET 不能缺省，而且每一行只能写一个语句。

2.2 True BASIC 的基本操作

True BASIC 程序编制完毕后，下一步工作就是将程序输入计算机，然后在计算机上修改、调试和运行。进行这些步骤时，需要正确地掌握操作方法和运用 True BASIC 所提供的各种操作命令。

2.2.1 上机操作步骤

上机操作的第一步是启动计算机，然后使计算机进入 True BASIC 工作状态。在已通电的微型计算机上实现这一过程的具体步骤是：

(1) 将带有操作系统和 True BASIC 翻译程序的软磁盘插入磁盘驱动器 A 中，然后关上驱动器门。插入磁盘时，必须使磁盘贴有标签的一面向上，磁盘读写孔朝前，小心插送，以确保磁盘与驱动器对位，再关驱动器门。如果插入操作不正确，极易损坏磁盘或驱动器。

(2) 同时按下[Ctrl]+[Alt]+[Del] 3 个键(操作时，先按住[Ctrl]、[Alt]两键，再按下[Del]键，然后同时放开 3 个键)，使计算机热启动(如果是在完成第 (1) 步后，再给主机加电称为冷启动)。热启动过程中，计算机将把磁盘上的操作系统读入计算机内存。读盘时，驱动器红灯发亮，表明驱动器正在工作，此时严禁移动磁盘！当计算机询问日期和时间时，可以分别按回车键([Enter] 或[Return]键)作回答。最后，屏幕上出现操作系统的提示符“A＞”，表明计算机启动完毕，已进入操作系统状态。

(3) 在 A＞的右边键入 Hello 并按[Enter] 或[Return]键，使计算机读取磁盘上的 True BASIC 翻译程序(程序名为 Hello.exe)，此时驱动器红灯再次发亮，数秒钟后，在屏幕的下方便显示出 True BASIC 设计者及版本号等信息，表明计算机已经进入 True BASIC 状态:

```
-----------------------------------------------------------
True BASIC here. Version 2.03
Copyright (C)  1988 by True BASIC,  Inc. All rights reserved.
Type SPLIT to see a menu of the function keys.
Ok.
```

其中，“Ok.”是 True BASIC 的提示符，表示计算机正在等待输入 True BASIC 的操作命令。

进入 True BASIC 状态后，整个屏幕被一横线划分为上下两部分。横线上部称为编辑窗口或程序窗口，用于书写程序和编辑文件。编辑窗口反映出计算机内存程序区中的内容。横线下部称为背景窗口或命令窗口，用于显示用户输入的 True BASIC 命令、程序的运行结果及出错信息。初次进入 True BASIC 状态后，光标总是首先出现在命令窗口上，可以直接键入 True BASIC 的各种操作命令。

如果计算机配置的是彩色显示器，那么，同时按下下述两键可以分别改变当前光标所在窗口的前景、背景和屏幕边缘的颜色:

[Ctrl]+[F]——改变当前光标所在窗口的前景颜色;

[Ctrl]+[B]——改变当前光标所在窗口的背景颜色;

[Ctrl]+[E]——改变屏幕边缘的颜色。

2.2.2 一些常用的操作键

两个窗口相互切换的功能键[F1]和[F2]:

按[F1]键，光标将移至编辑窗口。此时可以在编辑窗口内输入或修改程序。每输完一行，都应当按一次回车键([Enter] 或[Return]键)，以便光标移到下一行。按[F2]键，光标移至命令窗口并出现在最后一行的“Ok.”提示符右侧，这时可以输入 True BASIC 的各种命令。在命令窗口中每输入一行(不论是数据或命令)，也必须按一次回车键([Enter]或[Return]键)，表明本行输入结束。

以下是输入或修改程序时常用的编辑操作键:

〔↑〕、〔↓〕、〔←〕、〔→〕——上、下、左、右移动光标。

〔← Backspace〕——删除光标左侧的一个字符。

〔Del〕——删除光标位置上的字符。

〔Ins〕——取消或进入(光标变大或变小)插入字符的状态。

〔Shift〕——与其他键配合，用于输入一个键上的上档字符。

〔Caps Lock〕——切换字母大小写状态。

〔Num Lock〕——右边键盘一组键的光标数字功能切换开关。

2.2.3 操作实例

编辑和运行例 1 程序的操作过程如下：

(1) 进入 True BASIC；

(2) 按[F1]键将光标移到编辑窗口；

(3) 逐行输入程序；

(4) 按[F2]键将光标移回命令窗口；

(5) 键入 RUN 命令并按回车键([Return]或[Enter] 键)；

(6) 若程序输入无误，则命令窗口显示“?”，此时键入数字 2(圆半径值)并按回车键，计算机将计算出的圆面积 12.5664 显示在下一行，而后再现“Ok.”：

```
■ INPUT r
■ LET s=3.14159*r*r
■ PRINT s
■ END
------------------------
Ok. RUN
? 2
 12.5664
Ok.
```

(7) 如果程序输入有错误，光标将自动移至错误行，这时可用前述的编辑键修改程序；

(8) 重复(4)至(7)的步骤。

键入运行程序的命令 RUN 之后，计算机首先要对程序进行语法检查。如果发现错误则自动将光标移到程序的出错位置上，并在命令窗口中显示出错信息(参见附录 6)。此时，应立即纠正程序错误。改正一处错误后，再按一下[F2]键，光标就移至下一个错误的所在位置。如此反复，直至错误修改完毕，光标回到命令窗口为止。这时，可以再键入 RUN 命令使程序重新运行。

如果想清除内存中的程序，可以在“Ok.”之右边键入 NEW 命令，屏幕会出现提示：“Do you want to save this file?”询问是否需要存贮这个文件。此时按 N 键并回车表示不需存盘；如果按 Y 键并回车，则继续提示输入文件名(Enterfiles name as:)，从键盘上键入文件名，再按回车键即将当前程序以所给名字存入磁盘，并清除内存和屏幕上的程序。

命令窗口可以保留 50 行输入或显示的内容，可用[↑]键浏览以前输入或显示的内容，然后必须用[↓]或 [End]键返回到命令窗口的最后一行。 True BASIC 所有命令都必须在命令窗口最后一行的“Ok.”之右边输入。

2.2.4 几个常用的 True BASIC 命令

RUN

作用：运行编辑窗口(即内存)中的程序。

NEW

作用：清除内存及编辑窗口中的程序，如果程序预先未存盘则产生是否存盘的提示。凡准备输入一个新的程序之前，都应当使用一次 NEW 命令消除旧程序，以避免不同程序互相混淆。（若用 NEW myfile 回车，则 myfile 为即要输入的新文件名）。

SPLIT n

作用：改变屏幕上编辑窗口和命令窗口之间横线的位置。屏幕从上到下共有 25 行（第 25 行显示 True BASIC 提示符 Ok.），SPLIT n 命令可将横线移到第 n 行，n 必须为 0～24 之间的整数。例如：SPLIT 5 或 SPLIT 20，即编辑窗口占 5 行，命令窗口占（24～5）行；或编辑窗口占 20 行，命令窗口占（24～20）行。

以下命令与磁盘上程序的存取有关，在介绍这些命令之前，首先简述磁盘上程序的命名规则。

贮存在磁盘上的程序称为程序文件。每个程序文件必须有一个相应的文件名。文件名分为主名与扩展名两部分。主名最长允许 8 个字符，扩展名最长允许 3 个字符，主名与扩展名之间用句号“.”隔开。可用作主名或扩展名的字符有——

字母：A～Z

数字：0～9

特殊字符：$ # & @ ! % ( ) - { } ' '

例如：MyPrgm.1，DUDGET.%，C06013.TRU，PROGRAM9 均为合法的文件名。若文件名中省略了扩展名和句号，True BASIC 就默认其扩展名为 TRU，因此，PROGRAM9 与 PROGRAM9.TRU 在 True BASIC 中指的是同一程序文件。

FILES *.*

作用：显示当前磁盘上所有文件的目录。不带 *.* 的 FILES 命令则仅显示扩展名为 TRU 和 TRC（编译后的 True BASIC 程序文件的扩展名）的磁盘文件的目录。

OLD filename

作用：将当前磁盘上文件名为 filename 的程序调入内存，并在编辑窗口上显示出该程序。程序调入内存后可以运行或修改。

REPLACE filename

作用：将当前内存中的程序以指定的文件名 filename 存盘，以取代磁盘上原来存有的同名文件。如果省略文件名，则以取出该程序时所用的文件名存盘。此命令常用于将内存中的程序取出修改后再取代磁盘上原来的程序。

SAVE filename

作用：将当前内存中的程序以指定的文件名 filename 存入磁盘中。若磁盘上已存在与给定文件名同名的文件，则不能使用此命令，而必须用 REPLACE 命令。

UNSAVE filename

作用：删除磁盘上的指定文件 filename。磁盘上的文件被删除后，系统将收回文件所占用的存贮空间。文件一旦被删除，一般就无法再调出使用了。因此，使用这一命令时需要特别小心。

BYE

作用：退出 True BASIC 状态，返回到操作系统状态（即从 Ok. 返回到 A>）。

习 题 2

(1) True BASIC 程序设计语言有什么特点?

(2) True BASIC 程序是怎样构成的?

(3) True BASIC 有哪几种基本语句和结构控制语句?

(4) 什么是结构化程序设计，为什么说 True BASIC 语言是结构化程序设计语言?

(5) 如何书写 True BASIC 程序?

(6) 在 IBM PC 及兼容机上怎样进入 True BASIC 工作状态?

(7) 怎样输入并运行 True BASIC 程序?

(8) 用简单的基本语句和控制语句编写简单程序:

① 已知地球半径 R(6371 公里)，计算地球表面积($S=4\pi R^2$)。

② 地球陆地与海洋的面积之比为 3／10，世界总人口约53亿，计算世界平均人口密度(人／平方公里)。

③ 求中国各省人口密度(由键盘输入各省的面积和人口数)。

第3章　量与赋值

计算机处理的对象是“数据”。程序设计中涉及的数据有常量和变量两类。计算机对数据的运算处理反映为存贮单元中量值的变化，这种变化正是通过赋值来完成的。因此，量与赋值是计算机程序设计中两个最基本的重要概念。

3.1　数值常量与字符串常量

3.1.1　数值常量

在日常生活中，经常使用各种具体数字。例如：1，2，3，1001，2.25，-1.3，-178，+3.8，3.1415926，1990。这些具体的带正负号的十进制整数、小数是 True BASIC 最基本的数据形式。在计算机语言中，它们被称为数值常量。程序执行时，其量值保持不变。True BASIC 程序中，数值常量的写法与日常书写数字的方法是一致的。此外，数值常量还可以表示成指数形式（科学记数法），例如：

1E+3	（等同于 1×10^3 或 1000）
-2E-2	（等同于 -2×10^{-2} 或 -0.02）
1.989E+3	（等同于 1.989×10^3 或 1989）
3.75E-5	（等同于 3.75×10^{-5} 或 0.0000375）

指数形式的数由两部分组成：E 前面的数称为数字部分（或称为尾数）；E 后面的数称为指数（或称为阶码）。尾数为前述的数值常量，而指数为带符号的整数。注意 E 前必须有数字。

数值常量可由图 3-1 所示语法图描述。

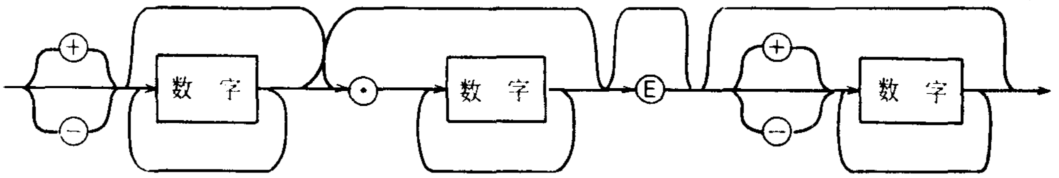
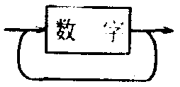
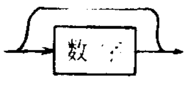


图 3-1　数值常量语法图解

其中“数字”为十进制的 0，1，2，3，…，9 数符之一；



表示可以重复；


表示可以绕过。

由数值常量语法图可见，数值常量由正负号、数字部分（可包括小数）和指数部分（如果是指数形式）组成。指数部分以字母 E（或 e）开头，后面跟一个位、二位或三位整数。如果数字部分（或指数部分）为正，则正号可省略。例如：

0.43，378，125.3，7E5，0.618e-2，3.31E+3，-11.7e-6，261.42，+2614.2E-1，+26.142E+1，2.6142E+2，2.6142e2，7980000000000，7.98E+12，-0.00000000000521，-5.21e-12

均为格式正确的数值常量。

对于绝对值很大或很小的数，用指数形式表示不仅可以书写得简洁，而且还可扩大数的表示范围。

尽管语法图未对数值常量中数字的位数作出规定，但对于具体的计算机而言，它所能表示的数却有一定的范围限制，因而是有限的。在普遍 IBM PC 机上，True BASIC 语言具有一个最大正数(1.79769E+308)和一个最小正数(5.56268E−309)。当所用数的绝对值大于最大正数时，计算机便无法表示这个数，程序运行时将出现"溢出"(Overflow)错误。如果数的绝对值小于最小正数，相对而言这个数就接近于零，此时 True BASIC 一律用零代替，而不显示出错信息。

True BASIC 处理数值数据一般具有 10 位有效位数(因计算机不同而有所差异)。对于有效数字超过有效位数的数值数据，True BASIC 将自动舍去尾部多出的有效数字。需要注意的是，尽管计算机内部按最高有效位数进行数据存储和运算，但在显示数据时，会自动转换成指数形式并且仅仅显示出有效数字的前 6 位。

3.1.2　字符串常量

计算机除了可处理数值信息外，还能够处理像姓名、地址、单词、词组、语句、文章等等由文字符号组成的信息。这种文字符号也是基本的计算机数据形式。

在计算机语言中，具体确定的文字符号序列称为字符串常量。True BASIC 语法规定字符串常量必须用一对双引号括起来。例如:

″This is a string″，″1990.6.1″，″m，n = ?″，″A+B = ″，″CHINA″，″Guilin College of Geology″，″Mark Twin″，″Tel: 121458″，″DN45−02″，″Please enter a leter: ″

以下是使用字符串常量的简单程序示例:

```
PRINT  ″I am a student.″
PRINT  ″My name is Wang Ping.″
PRINT  ″Computer science is very interesting!″
END
-----------------------------------
Ok.RUN
I am a student.
My name is Wang Ping.
Computer science is very interresting!
Ok.
```

True BASIC 的字符串常量必须符合语法图 3-2 的规定。

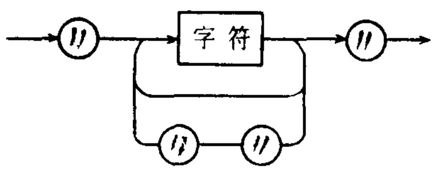


图 3-2　字符串常量语法图解

由图可见，字符串常量是包含在两个双引号(″ ″)之间的字符序列。括在字符串两端，起标识作用的双引号不属于字符串常量本身。字符可以由用户任意选定。如果字符串中包含双引号，那么作为字符内容的双引号应当双写，以便区别于括在字符串两端起标识作用的双引号。例如，把下面的整个英文句子

He said,"All right."

作为字符串常量时，就必须写成如下形式

"He said，""All right.""

一个字符串常量最多可以含有 32000 个字符。

在字符串常量中，英文字母大小写是有区别的。因此"ABC"与"abc"；"Joy"与"JOY"是不同的字符串常量。每一个字符都有一个固定的编码，称为 ASCII 码(美国标准信息交换码)。例如，大写字符 A 的 ASCII 码是 65，而小写字符 a 的 ASCII 码是 97。因此，由于 ASCII 码的不同，"A"与"a"是不同的字符串。应该特别指出，空格也是一个常用的字符，它的 ASCII 码是 32。不含任何字符的字符串称为空串，用内部不含字符的两个双引号(" ")表示。

字符串常量也可由数字构成，但此时作为字符串常量的数字并不具有数值意义，因而不能参与数值运算。以下两个简单程序说明了这种差别:

```
PRINT "3+4"                 PRINT 3+4
END                         END
--------------------        --------------------
Ok.RUN                      Ok.RUN
3+4                           7
Ok.                         Ok.
```

3.2 简单变量与下标变量

计算机处理的数据对象除常量外，更主要的是变量。变量是程序执行时其值可改变的有名字的数据对象。变量在数学中已为大家所熟知。程序设计语言中的变量与之基本相同，但又有差异。程序中的变量对应着计算机存贮器的某个存贮单元，存贮单元中的内容即为变量的值。存贮单元的内容在程序运行过程中是可以改变的，因此变量的取值是可变的。程序通过设置变量来使用计算机内部的存贮单元，进行内部状态的变换，从而达到信息处理的目的。

由于存贮单元既可存放数值数据，又可存放字符串数据，因此变量也分为数值型变量和字符串型变量。变量的取值范围与常量的取值范围相同。

True BASIC 中的变量分两大类：简单变量与下标变量。

3.2.1 简单变量

一个简单变量即相当于一个固定的存贮单元。在程序中设置变量，便可使用内存单元。简单变量是一种代表数据的符号。一个简单变量标识一个存贮单元，对单元中数据(数值数据或字符串数据)的运算处理，可通过对变量的操作来完成。

每个简单变量都有唯一的名称，称为变量名(或标识符)。简单变量的命名规则由变量命名规则图 3-3 表示。

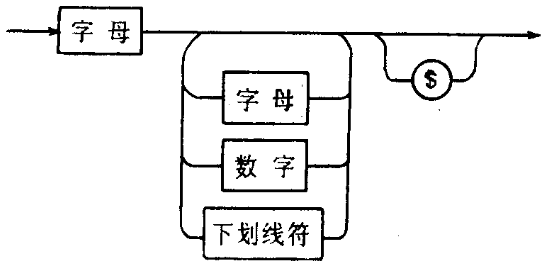


图 3-3 变量命名规则

变量名必须是以字母开头，后随字母、数字或下划线符(-)的符号串。其最大长度允许31个字符。在变量名中，大小写字母作同一符号对待，不予区分。如果变量中的内容是字符，则变量为字符串型变量。字符串型变量的变量名后必须带一美元符号($)。

以下为格式正确的变量名：

A，b1，ABC，B707，SUM，V3，R，S，Age，NAME，ADDRESS，Phone - No，P310．

为了使程序便于阅读和理解，在程序中为变量命名时，应根据数据的含义选取具有直观意义的变量名。如用R，I，V分别表示电阻，电流和电压；用Length，Width，Area分别表示长，宽和面积；用Name，Sex，age分别表示姓名，性别，年龄等等。

例如，已知电阻为500(欧)，根据电流(安)求出电压(伏)的程序可写成：

```
INPUT  I
LET  R = 500
LET  V = I * R
PRINT  V
END
```

3.2.2　下标变量

在科学研究、工程技术及事务处理中，经常使用一些成组出现的，具有一定排列次序的数据，如向量、矩阵、行列式、表格等等。在程序设计语言中，这类数据通常是用下标变量来表示的。

下标变量除具有变量名外，还具有下标。变量名的命名方法与简单变量相同。下标是跟在变量名后由圆括号标出的整数值。如果下标变量带多个下标，则下标之间用逗号隔开。例如：

A(1)，A(2)，Student(32)，Seat(3，30)，Seat(30，3)，City $ (20)

等均为合理的下标变量。变量名相同的一组下标变量，通常是作为一个数据整体来使用的(例如，表示一个矩阵、一张表格等)，因此，又称为数组。数组中下标变量的公共变量名又称为数组名。数组中的各个下标变量由不同的下标值来区别。但需要注意，True BASIC的下标变量不能与简单变量同名。

下标变量既可以是数值下标变量，也可以是字符串下标变量。由数值下标变量形成的数组为数值数组，由字符串下标变量形成的数组为字符串数组(字符串数组名后须具有一个 $ 符号)。

数组的维数是指变量所带下标的个数。例如：向量是一维的，向量的各个分量由一个下标区分；行列式、矩阵是二维的，其中每个元素的位置由所在的行和列确定，即变量带2个下标(行标和列标)。类似地，可以有3下标，4下标，…，乃至N下标的变量(IBM-PC机上的True BASIC，N最多等于255)。

每一维下标的最大、最小值称作该维的上界、下界。每维的上下界必须在 $-1 \times 10^9 \sim +1 \times 10^9$ 之间取值。

在使用下标变量(数组)之前，必须先用DIM语句说明数组名及每维的上下界。例如，以下程序首先建立一个具有3个下标变量的一维数组A，用以存放某学生3门课的成绩，然后求出平均成绩 M。

```
DIM A(1: 3)                             ---------------------
LET A(1)=83                             Ok. RUN
LET A(2)=78                             86
LET A(3)=97                             Ok.
LET M=(A(1)+A(2)+A(3))/3
PRINT M
END
```

在 DIM 语句中，说明数组各维的上下界时必须用常数，而不能用变量或表达式。

下标变量与简单变量的区别在于：下标变量的下标，允许用另一变量或表达式来表示，因此在编程序时具有通过下标选择变量的灵活性。例如，下面的赋值语句

```
LET A(i+2)=A(i)+A(i+1)
```

在 i=1 时，等效于

```
LET A(3)=A(1)+A(2)
```

在 i=2 时，等效于

```
LET A(4)=A(2)+A(3)
```

如果使用简单变量 Ai，则 Ai 是一个与 i 无关的独立的变量(关于下标变量的进一步讨论将在第 8 章中进行)。

在程序中每设立一个变量，计算机系统就开辟一个存贮单元来存放它的值。每设置一个数组，计算机就为此数值开辟一组存贮单元存放各下标变量的值。数值变量在设立时初值均为零，字符串变量在设立时初值均为空串。凡对变量赋值，都会“冲掉”(即“破坏”)变量原来的值，但引用变量则不会改变变量的值。

3.2.3 保留字

True BASIC 规定了一些保留字，其含义在系统中已事先有定义，因此用户不能把它们当做变量名使用。True BASIC 的下列 6 个关键字、11 个不带自变量的函数名和 3 个子程序名是被保留的。保留字如表 3-1 所示。

表 3-1 True BASIC 保留字

关键字	函数名		子程序名
ELSE	DATE	MAXNUM	DIVIDE
ELSE IF	DATE $	PI	PACKB
IF	DEF	RND	POKE
NOT	EXLINE $	TIME	
PRINT	EXTEXT $	TIME $	
REM	EXTYPE $		

如果使用了 True BASIC 的系统函数名作变量名，就无法再使用系统原来所规定的函数了。因此，实际上系统函数名也是被保留的。除表 3-1 列出的保留字外，凡符合变量名命名规则的字符组合均可以作为变量名。

3.3 赋　值

程序对数据进行操作变换，主要是通过对变量赋值来完成的。因此，赋值是程序设计中最基本最重要的操作之一。

3.3.1 赋值语句(LET)

True BASIC 对变量赋值的语句由关键字 LET 开始，它可以给一个(或多个)变量赋值。

语句格式:

```
LET  Var=expr
```

或

```
LET  Var1, Var2, ...=expr
```

作用：先计算出赋值号“ = ”右边表达式(expr)的值，然后把它赋给左边的变量(Var 或 Var1，Var2，...)。

如果赋值号右边表达式的值是数字，则左边的变量必须是数值变量；如果赋值号右边表达式的值是字符串，那么左边变量必须是字符串变量。即被赋值的变量类型应当与表达式的数据类型相匹配，例如:

```
LET  v=1/3*(Pi*R*R*H)       LET  NAME$="Firstname$&""&Surname$
LET  i, j, k=1              LET  P$"="End"
LET  Meter=50               LET  a$="TG"
LET  Million=1000*Meter     LET  e=2.71828
LET  Score1=75.5            LET  AVER=SUM/N
LET  Score2=84.3            LET  CAPITAL$="BEIJING"
LET  SUM=Score1+Score2      LET  A$="Begin"
```

3.3.2 赋值操作过程

特别需要指出的是，赋值语句不是普通的代数等式。虽然它们的外表形式相同，但其实质完全不同。赋值有它特定的操作过程，而代数式是一种关系概念。例如，下面的 3 个赋值语句:

```
LET  N=N+1,    LET  A=A*A-A,    LET  S=S+X
```

它们是符合语法规定的语句，但在代数中显然不存在或不讨论这种等式。

执行上述赋值语句的具体过程是:

(1) 将名字为 N 的变量所对应的存贮单元中的数值与常数 1 相加，然后将结果置于该变量单元中。赋值的结果使对应变量 N 的单元中的数值加了 1。

(2) 将对应变量 A 的存贮单元中的数值读出来，作自乘运算后，又从乘积中减去所读出的数值，把结果再存入 A 所对应的单元中。完成赋值处理后，原 A 单元中的旧值已为新值所取代。

(3) 将对应变量 S 和 X 的存贮单元的内容读出，相加后再置入变量 S 的单元中。赋值结果 X 单元的值不变，而 S 单元中存入了新值，旧值已不复存在。

另外，在代数关系中，$A=x+y$ 和 $x+y=A$ 是等价的，但 True BASIC 的赋值语句却不允许写成 LET x+y=A。因此，必须特别注意，不要简单地将代数方程的求解直接用赋值语句来表达，而应当将方程求解的算法步骤用一系列符合语法规则的语句来完成。

实际上，赋值语句反映的是一个数据运算与传递的动态过程，并不是代数中抽象的静态恒等关系。在赋值语句中，假如赋值号左右两侧出现相同的变量，那么右边的变量表示本次赋值前的值，而左边的变量表示本次赋值后的新值。初学者必须弄清这种差别。

3.3.3 几个程序段

计数、累加、累乘以及两个变量内容互换的程序，是编程中常用的程序段。这些程序段都借助了赋值语句来实现其功能。

程序 1 记录循环次数并显示次数值的平方与立方，当循环达到 100 次时退出循环。

```
LET  i=0
DO
     LET  i=i+1
     PRINT  i, i*i, i*i*i
     IF  i=100  then
         EXIT  DO
     END  IF
LOOP
END
```

其中的 LET i=i+1 语句，对每次循环作加 1 处理，起计数作用。一般的计数形式程序段为:

```
LET  i=0            ! 初始化
DO
    ……
    LET  i=i+1      !计数
    ……
LOOP
```

程序 2 不断从键盘输入数据，作求和运算。

```
LET  s=0
DO
   INPUT  x
   IF  x=0 then
       EXIT  DO
   END  IF
   LET  s=s+x
   PRINT  s
LOOP
END
```

其中，LET s=s+x 语句即起累加作用，它将每次输入的 x 的数值加入求和单元 s 中。一般的累加形式程序段为:

```
LET  s=0            !初始化
DO
      ……
      LET s=s+x     !累加
      ……
LOOP
```

程序 3　计算 20 的阶乘数(即 1 × 2 × 3 × … × 19 × 20)。

```
LET   t=1
LET   n=0
DO
    LET n=n+1
    LET t=t*n
    IF n>20 then
       EXIT DO
    END IF
LOOP
PRINT t
END
```

其中，LET t=t*n 语句即起累乘作用。由于 n 为计数变量，每循环一次 n 就加1。因此，循环执行 LET t=t*n 时，n 的不同取值便被乘入 t 中。当 n 大于 20 时，1，…，20 已乘入 t。此时循环结束，程序输出 t 的值。一般的累乘形式的程序段为：

```
LET t=1               !初始化
DO
    ……
    LET t=t*n         !累乘
    ……
LOOP
```

程序 4　使两个变量内容相互交换。

```
LET   aS ="W-203"
LET   b$ ="Z-311"
PRINT  a$, b$
LET   t$ =aS
LET   aS =bS
LET   b$ =t$
PRINT  a$, b$
END
```

一般，两个变量内容的交换，需经过

```
LET   t$ =a$               LET   t=x
LET   a$ =bS       或       LET   x=y
LET   b$ =t$               LET   y=t
```

的形式完成。

习　题　3

(1) 计算机中对量的表示与数学中的量有什么异同点？

(2) 何谓数值的“溢出”？

(3) 试说明简单变量与下标变量的主要区别?

(4) 赋值符号“=”与代数等于号“=”有何不同?

(5) 指出下列符号中正确的数值常量、字符串常量、简单变量、下标变量、数值变量、字符串变量:

11.4E12　″-30.4″　0　TrueBASIC　2.7×10^7　±0.1　π　$123　6abc　AbC$
2.1(0)　GO_TO　″A(2)″　4.5E_0.1　2+3　10%　ELSE　IF$(1)　E10　Name$(1)
Tom　″1.1(1)″　0.00001　K2$　″ELSE″　″A″$　E-2　2.87E2.81　.01　4.5×E3
PRINT　No.1　″-2E-2″　A123　IMB-PC

(6) 按下列公式写出 True BASIC 赋值语句:

① $V=4\pi R^3/3$　② $Z=X^2+Y^2$　③ $S=V_0t+gt^2/2$

(7) 指出下列赋值语句中的错误并改正:

①LET a, b, c=1, 2, 3　②LET A=12　③LET B=A=C
④Name='Tom'　⑤ One=2　⑥ LET Words=″Please answer 'yes' or 'no': ″　⑦LET M=N A=B　⑧LET A=1 LET B=2

(8) 写出执行以下程序段后各变量的值:

```
①LET a=10
  LET b=15
  LET b=b+a
  LET a=b
  LET c$=″a+b″
```

```
②LET a=1
  LET b=2
  LET a=a+b
  LET b=a-b
  LET a=a-b
```

```
③LET a=b+c
  LET c=b-c
  LET c=a+c-2*b
  LET b=c*2+1
  LET a=2*b
```

(9) 写出计算下列公式的程序:

① $\sum_{n=1}^{30} 2^{-n}$　② $\prod_{n=1}^{11}\frac{2n}{2n-1}$　③ $\sum_{n=0}^{10} n!$

第 4 章　函数和表达式

本章讨论程序设计中进行数值计算、字符和时间处理时各种常用函数的表达方式及运算求值的规则。True BASIC 提供了两大类函数：系统函数和自定义函数。这一章介绍系统函数。自定义函数将在第 9 章讨论。

4.1　系统函数

系统函数(也称为标准函数)是 True BACIC 系统本身带有的函数。一些形式比较简单的数值函数、三角函数、字符串函数等，由于它们在一般数值运算及文字处理中使用很频繁，因此在设计 True BASIC 语言时就作为一种功能置于该语言系统之内。每个函数既代表自变量进行计算的规则，又给出计算的结果。使用函数时只要写出函数名并在圆括号内给定自变量，即可获得函数值。

4.1.1　数值函数

这一类函数的自变量和函数值均为数值。下面说明函数时，除特别注明，x，y 和 n 可以是任意数值或数值表达式。调用是指将自变量的数值代入该函数，返回是指输出的函数值。下面每说明一个函数后，即举出几例。例中左边是函数调用，右边是调用的返回值。

ABS(x)　返回 x 的绝对值(即 $|x|$)。

调用	返回
ABS(77.3)	77.3
ABS(0)	0
ABS(−35.2e4)	352000

EXP(x)　返回 e^x 的值(它的逆运算是 LOG 函数)。

调用	返回
EXP(0)	1
EXP(1)	2.71828
EXP(13.3)	597196

INT(x)　返回一个小于或等于 x 的最大整数。因此,若 x 是一个整数,则 INT(x) = x. 其他情况下，INT(x)将返回不大于 x 的最大整数。

调用	返回
INT(300.9)	300
INT(−1.2)	−2
INT(3.5)	3

LOG(x)　返回 x 的自然对数(即 $\ln x$，EXP 是 LOG 的逆运算)。因为负数或 0 没有对数，因此当 $x<0$ 时，机器系统将给出出错信息(附录 6)。

调用	返回
LOG(1)	0
LOG(EXP(10))	10

LOG2(x)　返回 x 的以 2 为底的对数(即 $\log2x$)。例如:

调用	返回
LOG2(1)	0
LOG2(4)	2
LOG2(2^10.1)	10.1
LOG2(0.5)	−1

LOG10(x)　返回 *x* 的以 10 为底的对数(即 lg*x*)。

调用	返回
LOG10(10)	1
LOG10(1e25)	25
LOG10(.001)	−3

MAX(x，y)　返回所给 *x*，*y* 中的较大数.

调用	返回
MAX(1，3)	3
MAX(1，−1)	1

MAXNUM　返回机器上所能使用的最大正数(这个数在不同的机器上有所不同)。对于 IBM−PC 机，MAXNUM 给出的最大值是:

调用	返回
MAXNUM	1.79769e+308

MIN(x，y)　返回所给 *x*，*y* 中的较小数.

调用	返回
MIN(1，3)	1
MIN(1，−300)	−300

MOD(x，y)　返回“*x* 模 *y*”的值。即 *x* 除以 *y* 的余数。因为 0 不能做除数，所以当 *y* 为 0 时，True BASIC 将给出一个出错信息(附录 6)。求模运算可由下式表达：MOD(x，y)＝x−y * INT(x／y)，其结果在 0 与 *y* 之间。

调用	返回
MOD(5，2)	1
MOD(5.3，2)	1.3
MOD(4，4)	0
MOD(−1，3)	2
MOD(7，−2)	−1

RND　返回一个伪随机数，并且 0<Rnd<1。尽管机器并不能计算出真随机数，但 Rnd 返回的数无明显规律。每次运行程序，True BASIC 都给出相同的伪随机数序列。如果在使用 Rnd 之前执行一条 RANDOMIZE 语句，那么，Rnd 函数将得到不同的伪随机数序列。

调用	返回
RND	.55337
RND	.022
RND	.32561
RND	.83856

ROUND(x) 返回 x 经过四舍五入后的整数，它等价于函数 Int(x+0.5)。

调用	返回
ROUND(1.23)	1
ROUND(1.6666)	2
ROUND(−3.2)	−3

ROUND(x，n) 返回 x 取 n 位小数后四舍五入的近似值。若 $n>0$，结果近似到小数点右边第 n 位。$n<0$，则近似到小数点左边第 n 位。

调用	返回
ROUND(112.237，2)	112.24
ROUND(112.237，1)	112.2
ROUND(112.237，0)	112
ROUND(112.237，−1)	110
ROUND(112.237，−2)	100

SGN(x) 返回自变量 x 的符号。如果 $x>0$ 返回 1；$x=0$ 返回 0；$x<0$ 返回−1。

调用	返回
SGN(137)	1
SGN(0)	0
SGN(−1.2e−2)	−1

SQR(x) 返回 x 的平方根。因为负数没有实数方根，故 x 为负时，True BASIC 将给出出错信息。

调用	返回
SQR(2)	1.41421
SQR(100)	10

例 1 已知 $p(x，y)$和 $p_0(x_0，y_0)$点的平面坐标值，利用开平方函数计算并输出两点之间的距离$\left(r=\sqrt{(x-x_0)^2+(y-y_0)^2}\right)$。

程序:

```
INPUT  x0，y0，x，y
LET  r=sqr((x−x0) * (x−x0)+(y−y0) * (y−y0))
PRINT  r
END
```

例 2 已知 r_0，t，h，利用指数函数和自然对数函数计算并输出 r。

$$r=r_0e^{-(\ln 2t/h)}$$

程序:

```
INPUT r0，t，h
LET r=r0 * exp(−log(2) * t / h)
PRINT r
END
```

例 3 在某种比赛中，假设有 10 位裁判评分。计算分数时规定，扣除一个最低分和一个最高分，其余的分数取平均值即为最后得分。

程序:

```
LET  s, i=0
DO
   LET  i=i+1
   INPUT  x
   LET  s=s+x
   IF  i=1  then
      LET  mx, mn=x
   END  IF
   IF  i>1  then
      LET  mx=max(mx, x)
      LET mn=min(mn, x)
   END IF
   IF i=10 then
      EXIT DO
   END IF
LOOP
LET s=(s-mx-mn)/8
PRINT s
END
```

例 4 已知某日为星期天，利用求余函数 MOD 回答此日后的第几天为星期几?
程序:

```
DIM  w$(0:6)
LET  w$(0)=″Sun.″
LET  w$(1)=″Mon.″
LET  w$(2)=″Tues.″
LET  w$(3)=″Wed.″
LET  w$(4)=″Thu.″
LET  w$(5)=″Fir.″
LET  w$(6)=″Sat.″
DO
   INPUT n
   IF n=-99 then
      EXIT DO
   END IF
   LET r=mod(n, 7)
   PRINT w$(r)
LOOP
END
```

例 5 某学生每天早晨起床约需 7～10 分钟，从宿舍走到食堂 3 分钟，排队买饭 0～5 分钟，吃饭 6～8 分钟，走到教室 4 分钟。作一程序统计该学生 *n* 天的从起床至到达教室的平均时间。

问题的关键在于求出起床、排队和吃饭时间的统计数。用随机函数 RND 模拟这 3 个时间:

t1=7+round(3 * rnd)； t2=round(5 * rnd)； t3=6+round(2 * rnd)。

程序:

```
INPUT  n
LET  t0=3
LET  t4=4
LET  t1, t2, t3=0
LET  i=0
RANDOMIZE
DO
   LET  t1=t1+(7+round(3 * rnd))
   LET  t2=t2+round(5 * rnd)
   LET  t3=t3+(6+round(2 * rnd))
   LET  i=i+1
   IF  i>n  then
      EXIT  DO
   END  IF
LOOP
LET  t=t0+(t1+t2+t3)/n+t4
PRINT ″Mean=″; t
END
```

4.1.2 三角函数

三角函数中所用的角度一般是用度表示，而在 True BASIC 中当不作说明时，是用弧度

表示的。

若在程序中有一条 OPTION ANGLE DEGREES 语句，则其后的三角函数的自变量和结果都将以度为单位；而另一条 OPTION ANGLE RADIANS 语句，则会使所有的角都以弧度表示。Deg 和 Rad 函数不遵循这条规则。OPTION 语句应出现在使用三角函数之前。

ATN(x)　返回 x 的反正切值。

调用	返回
ATN(0)	0
ATN(pi)	1.26263

COS(x)　返回 x 的余弦值。

调用	返回
COS(0)	1
COS(pi)	−1

DEG(x)　返回把 x 弧度转化为角度后的值。

调用	返回
DEG(pi)	180
DEG(pi／4)	45

PI　返回 π 的值(3.1415926535...)，精确到该机能处理的位数。

RAD(x)　返回把 x 度转化为弧度后的值。

调用	返回
RAD(180)	3.14159 (即 π)
RAD(45)	0.785398 (即 $\pi/4$)

SIN(x)　返回 x 的正弦值。

调用	返回
SIN(0)	0
SIN(Pi／2)	1

TAN(x)　返回 x 的正切值。

调用	返回
TAN(0)	0
TAN(Pi／4)	1

例 6　根据直角三角形两直角边 a 和 b，计算并输出角 B 的正弦、余弦、正切值。

程序：

```
INPUT  a，b
LET  c=sqr(a*a+b*b)
LET  bsine=b/c
LET  bcosine=a/c
LET  btangent=b/a
PRINT  bsine，bcosine，btangent
END
```

例 7　利用标准涵数计算并输出 100～120 度之间(间隔 1 度)的正弦、余弦、正切值。

程序:

```
OPTION  ANGLE  degrees
PRINT  ″x″, ″sinx″, ″cosx″, ″tanx″
LET  x=100
DO
    PRINT  x, sin(x), cos(x), tan(x)
    LET  x=x+1
    IF  x>120  then
        EXIT  DO
    END  IF
LOOP
END
```

4.1.3 字符串函数

这些函数以字符串或函数值作为自变量，当返回值为字符串时，函数名后带有 $ 。在下面的函数中，除特殊注明外， a$ 和 b$ 都是字符串表达式， n 为数字表达式。

CHR$(n)　以一个数字作为自变量，返回对应于这个数码的字符。字符代码 0 至 127 在 ASCII 字符集中，并且在所有的机器上都相同。而代码 128 至 255 在不同的机器上有所不同。若所使用的数码超出 0 至 255 的范围，则 True BASIC 采用 Mod(n, 256)的方法把该数限定在这个范围以内。

调用	返回
CHR$(65)	″A″
CHR$(97)	″a″
CHR$(32)	″ ″
CHR$(65+256)	″A″

LEN(a$)　以字符串为自变量，返回这个字符串的长度(即包括空格在内的字符个数)。

调用	返回
LEN(″THREE″)	5
LEN(″AMERICA, TEXAS)	13
LEN(″″)	0

LCASE$(a$)　返回把 a$ 中的大写字母改成小写后的 a$ 。

调用	返回
LCASE$ (″WYATT EARP″)	wyatt earp
LCASE$ (″MARAINEY″)	marainey
LCASE$ (″++P.T.BARNUM++″)	++p.t.barnum++

LTRIM$(a$)　以一个字符串为自变量，把这个串开头的空格修整掉。

调用	返回
LTRIM$(″OHO ″)	″OHO″
LTRIM$(″I SEE″)	″I SEE″
LTRIM$(″″)	″″

ORD(a$) 以一个字符为自变量，返回该字符之字符代码(ORD 的逆运算是 CHR$)。在 ASCII 字符集里，定义了 128 个字符，在任何机器上，它们的代码都一样。

调用	返回
ORD (″A″)	65
ORD (″a″)	97

POS(a$,b$) 用来查找 a$ 中第 1 个与 b$ 相匹配的内容,并且返回第 1 个匹配内容的第 1 个字符的位置。如果没有找到, 则 POS 返回 0。

调用	返回
POS(″abcdef″, ″c″)	3
POS(″abcdef″, ″def″)	4
POS(″abcdef″, ″cc″)	0

POS(a$, b$, n) 也用来查找 a$ 中与 b$ 相匹配的内容，并且返回 a$ 中从第 n 个字符位置起的第 1 个与 b$ 相匹配内容的第 1 个字符的位置。若开始字符次第号 n 小于 1，则 Treu BASIC 用 1 代替。若大于字符串的长度或者无匹配串， POS 返回 0。

调用	返回
POS(″willy-nilly″, ″ill″, 2)	2
POS(″willy-nilly″, ″ill″, 3)	8
POS(″willy-nilly″, ″ill″, 30)	0
POS(″willy-nilly″, ″silly″, 1)	0

REPEAT$ (a$, n) 以一个字符串和一个数作为自变量，把该字符串重复 *n* 次。

调用	返回
REPEAT $ (″x″, 5)	″xxxxx″
REPEAT $ (″ox″, 3)	″oxoxox″
REPEAT $ (″x″, 0)	″″

RTRIM$(a$) 返回把 a$ 的结尾处空格修整掉的结果。

调用	返回
RTRIM $ (″ splendid ″)	″splendid″
RTRIM $ (″No Nonsense. ″)	″No Nonsense.″
RTRIM $ (″ ″)	″″

STR$(N) 把数 N 转为字符串的形式。

调用	返回
STR $ (3)	″3″
STR $ (3 + 0.14)	″3.14″
STR $ (.0000000009)	″9.e-10″
STR $ (2 - 5)	″-3″

TRIM$(a$) 返回把这个 a$ 的开头空格和结尾空格都删掉后的 a$ 。

调用	返回
TRIM $ (″ Niihau Maui″)	″Niihau Maui″
TRIM $ (″ Molokai ″)	″Molokai″
TRIM $ (″ ″)	″″

UCASE $ (a $)　返回把 a $ 中所有小写字母改为大写字母后的 a $ 。

调用	返回
UCASE$("Discover")	"DISCOVER"
UCASE$("The First State")	"THE FIRST STATE"
UCASE$(" CT 17304 ")	"CT 17304"
UCASE$(" price= $10.95")	"PRICE= $ 10.95"

USING$(format$，expr1，expr2，...)　返回一个将表达式按规定格式连接而成的字符串。其中，format$为描述字符串(见 5.2.3)。

调用	返回
USING $ ("#:####"，PI)	3.1416
USING $ ("# is less than #."，4，6)	4 is less than 6.

VAL(a $)　返回一个数。这个数是把 a S 转换为数字的结果。若这个串不能表示成正确的数，则 Ture BASIC 给出出错信息。字符串的开头和结尾空格将被忽略(Val 的逆运算是 Str $)。

调用	返回
VAL("3.1415")	3.1415
VAL("1.945e +3 ")	1945
VAL("-33.　　")	-33

4.1.4　时间函数

TIME　返回当前的时间(以秒为单位)。

调用	返回
Time	365
Time	501

例 8　按照计算机的提问进行简单对话。

程序:

```
PRINT   "Are you a student or a teacher ?"              ! 屏幕提问
INPUT   nswer $                                         ! 输入回答(一个单词)
LET   a $ =ucase $ (answer $ )                          ! 将回答的单词变成大写，以便
IF   a $ ="STUDENT"   then                              ! 与"STUDENT"、"TEACHER"
  PRINT   "You must study hard !"                       ! 进行比较。
END   IF
IF   a $ ="TEACHER"then
  PRINT   "I want to be a teacher, too !"
END IF
IF pos("STUDENT TEACHER", a $ )=0  then                 ! 回答的单词不在单词表内
  PRINT   "Sorry, I don't know what you said."
END IF
END
```

例 9　分别记录计算机重复 1 万次判断及加法的大概时间和 1 万次乘法的大概时间。

程序:

```
LET  n=10000
LET  i=0
LET  t0=time
DO
     LET  i=i+1
     IF  i>n  then
        EXIT  DO
     END  IF
LOOP
LET  t1=time
LET  i=0
DO
     LET  i=i+1
     LET  m=pi*pi
     IF  i>n  then
        EXIT  DO
     END  IF
LOOP
LET  t2=time
LET  tk=t1-t0          ! 1万次判断及加法时间
LET  tm=(t2-t1)-tk     ! 纯1万次乘法时间
PRINT  tk, tm
END
```

4.2 表达式

表达式是一个或多个运算的组合式，是运算求值的规则，是对数据进行处理和构造逻辑判断条件的重要手段。True BASTC 将表达式分为 3 大类：数字表达式；字符串表达式；逻辑表达式。

4.2.1 数字表达式

由数字常量、数值变量、数值函数和算术运算符(加、减、乘、除、乘方)按规则构成的运算式，称为数字表达式。在表达式中可使用圆括号来控制计算的次序。

算术运算符(表 4-1)具有不同的“优先级”，在同一级内的运算由左到右。括弧最优先 (其顺序是先里层后外层)；第 2 级是函数计算；第 3 级是幂运算；第 4 级是乘除法；第 5 级是加减法。注意，0^0 定义为 1；零不能作除数；负数无对数；乘号“*”不能省略；除号“/”是左斜杠；括号应成对出现。总之，要使表达式能正确地反映运算的原意。

在表达式的计算过程中，运算的精确度会因机器不同而有所差异。True BASIC 计算表达式时至少保留 10 位数，但在输出显示时仅为 6 位有效数。

表 4-1　算术运算符

算符	含　义
+	加法(或正号)
−	减法(或负号)
*	乘法
/	除法
^	幂
( )	括号(可以嵌套使用)

4.2.2 字符串表达式

对字符串常量、字符串变量和字符串函数，使用字符串算符&可以构成字符串表达式。对字符串可进行两种运算：字符串合并(&)与字符串提取。用运算符 & 可以把两个字符串连接在一起；使用〔a：b〕可获得字符串表达式的子串，子串的起始和结束位置分别由数 a，b 确定。两种运算符同时出现时，字串提取优先于字符串合并。例如：

若 A$ = “100”；B$ = “05”；C$ = “.”，则 A$&C$&B$ 的运算结果为 “100.05”。

“Micro”&“soft”&“ BASIC”的运算结果为 “Microsoft BASIC”，

“True ”&“BASIC”运算结果为 “True BASIC”。

“ABC”&“.XYZ”[2:4] 运算结果为 “ABCXYZ”。

(“ABC”&“.XYZ”)[2:5]，运算结果为 “BC.X”。

例 10 从键盘输入一个正整数，利用字符串函数及取子串运算，将数值的各位数字分离出来并计算出它们的和。例如，数值 725，分离后各位数字之和为 7+2+5＝14；数值 341，其各位数字之和为 3+4+1＝8，等等。

程序：

```
INPUT  n                                  ! 输入正整数(少于7位)
LET  p$ =str$(n)                          ! 转换成字符串
LET  i, s=0
DO
     LET  i=i+1
     LET  a$ =p$ 〔i：i〕                  ! 提取p$的第i个字符
     LET  s=s+val(a$)                     ! 将该字符转换成数字并累加进s中
     IF  i>len(p$)  then                  ! 超过字符串长度则退出
         EXIT  DO
     END  IF
LOOP
PRINT  s                                  ! 显示和数s
END
```

程序中将数值转换成字符串，然后从中提取某个字符，再转换成数进行处理。这种方法对处理一个数据的某一位是可借鉴的。

4.2.3 逻辑表达式

由关系表达式、逻辑算符、逻辑函数等组成的表达式，称为逻辑表达式。逻辑表达式常用于判断结构和循环结构中，作为判定条件使用。逻辑表达式的值只有真(True，即条件成立)、假(False，即条件不成立)值。

(1)关系表达式 是用关系运算符(表 4–2)连接的两个算术表达式(或字符串表达式)。

关系运算符只能对两个数字表达式或两个字符串表达式作运算，而不能对数字与字符串的混合式作运算，即比较只能在同类表达式间进行。字符串的比较方法采用字典方式，以 ASCII 码为序。注意：空格也是一个重要字符，空串小于任何有字符的字符串。每个关系表达式，当其中的变量确定时，都有其真假值。例如：

3＞5	其值为假(False)
″A″＞″a″	其值为假(False)
p$ ＝q$	其值由 p$ 是否等于 q$ 决定
″a″＜″b″	其值为真(True)
2＝2	其值为真(True)
MOD(n，7)＝0	其值当 n 为 7 的倍数时为真，否则为假
a＞b	其值依变量 a，b 的具体数值而定

(2)复合表达式 当需要构造复合关系式时，必须使用表 4–3 逻辑运算符。

表 4-2　关系运算符

算　符	含　义
<	小于
<=或=<	小于等于
=	等于
>	大于
>=或=>	大于等于
<>或><	不等

表 4-3　逻辑运算符

算　符	含 义
NOT	逻辑非
AND	逻辑与
OR	逻辑或

逻辑运算符优先级次序为: NOT, AND, OR。圆括弧可用来控制运算顺序。逻辑运算符的大小写不作区分。例如:

复合关系式	真　　值
A=B　OR　C=D	由 A, B, C, D 的值决定, 只要其中一个等式成立则表达式的逻辑值为真
A=B　AND　C=D	由 A, B, C, D 的值决定, 只有两个等式同时成立, 表达式才取真, 否则为假
A$<B$　OR　B$<A$	表达式值永真
NOT (A=B and C=D)	当括号内的逻辑表达式为真时, 整个表达式取假, 反之为真

又例如, 当需要表达数学条件　0<x<=A　时, 这种复合关系应写成:

x>0　AND　x<=A

而表达数学条件　y<−1 或　y>1　时, 应写成:

y<−1　OR　y>1

表达条件　1<x<3 或 −3<x<−1　则应写成:

(x>1 AND x>3) OR (x>−3 AND x<−1)

(3) **逻辑函数**　逻辑表达式中可以包含函数值只取真假值的逻辑函数。例如:

KEY INPUT	当键盘上有一个输入时(即按了任一键)其值为真
END DATA	若无数据项等待 READ 语句读时, 其值为真, 否则为假
MORE DATA	若有数据项等待 READ 语句读时, 其值为真; 否则为假
END #expr	若到达文件#expr 的结尾, 其值为真, 否则为假
MORE #expr	若在文件 #expr 的开始或中间, 其值为真, 否则为假

关于后 4 个逻辑函数的使用, 将在以后的章节介绍。

例 11　循环显示时间, 直至按任一键后程序停止(利用 KEY INPUT 函数)。

程序:

```
DO
    LET  t=time
    LET  h=mod(int(t/3600), 24)                        ! 计算小时数
    LET  h=mod(int(t/60), 60)                          ! 计算分钟数
    LET  s=mod(t, 60)                                  ! 计算秒数
    PRINT  USING$("%%:%%:%%", h, m, s)                 ! 以"小时: 分钟: 秒"的形式显示
    IF  key  input  then                               ! 按任一键则退出
        EXIT  DO
    END  IF
LOOP
END
```

习 题 4

(1) 把数学条件：$X \in (5, \infty)$或$(-\infty, -5)$且$Y \in (2, \infty)$写成对应的 True BASIC 条件表达式。

(2) 将下列数学表达式改写成对应的 True BASIC 表达式：

①X^2Y^2　　②$(X \cdot Y)^a$　　③$n \times (n+1) + k$

④$\sqrt{ab-cd}$　　⑤$\dfrac{\log_{10}{}^{23}}{\ln{}^{18}}$　　⑥$\dfrac{\sin x}{|\cos 15^\circ|}$

(3) 改正下列表达式中的错误：

①2X，　②a * sinX，　③X╲Y，　④A $ +B $，　⑤3.14 / −2.1 。

(4) 利用求余函数写出某月第 1 日是星期 M，求该月第 N 日是星期几的算法。

(5) 已知：$B=3$，$X=0.8$，$A=2$，$\omega=1.5$，$\varphi_0=0.3$，求阻尼振动曲线

$$Y = B \times \mathrm{e}^{-AX} \sin(\omega X + \varphi_0)$$

的函数值。

(6) 绝对温度为 T 的黑体，对波长为 λ 的幅射功率为：

$$E = \frac{2\pi ch\lambda^{-5}}{\mathrm{e}^{\mathrm{ch}} / \beta\lambda^{T} - 1}$$

其中，光速 $c = 2.997924 \times 10^8$；普朗克常数 $h = 6.6252 \times 10^{-34}$；波尔兹曼常数 $\beta = 5.6687 \times 10^{-8}$。编写一程序读入 T 和 λ 的值，计算并输出 E 的值。

(7) 已知四位数 A，编写程序段将 A 的后两位与前两位颠倒构成一个新的四位数 B。

(8) 编程计算进行 1 万次纯粹的加、减、乘、除及乘方运算所需要的时间，并判断在计算 R^2 时应选择(　)：①R * R；②R^2；③EXP(2 * LOG(R))。

第5章　输入和输出

在第2章开头已提到，程序一般都包括有输入(Input)、处理(Processing)和输出(Output)三部分。输入是为计算机处理提供数据；处理是对数据进行加工变换；而输出则是把运算处理的结果显示、打印或送给其他外部设备(如磁盘、磁带等)。因此，输入输出是程序设计的重要内容。正确地理解和掌握数据的输入／输出技术，才能有效地完成信息处理的任务。

向计算机主机输入的数据，主要来源于:

(1) 从计算机终端键盘上输入的数据；

(2) 随同程序的 Data 语句装入内存数据区的数据；

(3) 作为计算机外存贮器的磁盘、磁带上的数据文件。

类似于输入，主机也能向一系列外部设备输出经过处理的数据。一般包括:

(1)向计算机终端屏幕输出；

(2)向行式打印机输出；

(3)向磁盘、磁带输出。

5.1　INPUT 语句

数据输入方法中，使用 INPUT 语句，通过键盘把数据提供给程序变量，是最普通、最方便灵活的一种。这种交互式的人机对话方法，使操作者可通过键盘与计算机程序往复交换信息。

5.1.1　INPUT 语句的格式

由键盘输入数据的语句一般格式:

INPUT　Var1，Var2，…，VarN

功能：当执行 INPUT 语句时，程序将暂停并显示问号“?”，以等待操作者从键盘上为n个变量 Var1，Var2，…，VarN 提供n个数据。

其中，Var1，Var2，…，VarN 既可以是数值变量，也可以是字符变量。为它们提供的数据的类型和个数必须相互匹配。例如:

INPUT　a　要求在命令窗口的? 号后输入1个数值数据，然后按回车键。

INPUT　a，b　要求输入2个数值数据，数据中间应以逗号分隔，然后按回车键。

INPUT　P$，X　要求输入1个字符串和1个数字，中间以逗号分隔，然后按回车键。

INPUT　L$，R$，M$　要求输入3个字符串，它们中间以逗号分隔，然后按回车键。

5.1.2　带提示串的 INPUT 语句

语句格式:

INPUT　prompt　expr$：Var1，Var2，…，VarN

功能：与5.1.1节所述基本相同. 差别仅在于这种带提示串 expr$ 的 INPUT 语句被执行时，程序暂停并显示 expr$ 的内容作为输入提示(不再显示问号)，然后等待用户从键盘输入相应的数据。其中，作提示用的字符串表达式 expr$，是由用户任意给定的提示信息。

例 1　根据任意三角形的 3 条边 a，b，c，计算三角形的面积 S:

$$S=\sqrt{p(p-a)(p-b)(p-c)}$$

其中，$p=(a+b+c)/2$。

程序:

```
INPUT a, b, c                                    !从键盘上输入 a, b, c 的数值
LET  p=(a+b+c)/2
LET  s=sqr(p*(p-a)*(p-b)*(p-c))
PRINT  s
END
```

程序运行到执行 INPUT 语句时，屏幕上将显示问号“?”。当操作者在其后键入 3 个以逗号“,”分隔的数据并按回车键，程序便向下继续执行，计算 p，S 的值，并输出 S 的值:

```
-----------------------------
Ok. RUN
? 18, 15, 27
 127.279
Ok.
```

如果将程序的第一个语句改为带提示串的形式:

```
INPUT  prompt "a, b, c=?": a, b, c
```

再运行程序时，屏幕上将出现上面语句中所给的提示字符串。按此提示输入数据后，程序继续执行，输出结果:

```
--------------------
Ok. RUN
a, b, c=? 18, 15, 27
 127.279
Ok.
```

例 2　输入某人的姓名、性别和年龄，根据婚姻法规定，判断他(她)是否符合法定结婚年龄(男性大于等于 22 周岁，女性大于等于 20 周岁)。

程序:

```
DO
   INPUT  prompt "Name, Sex(""f""or""m""), Age=?": n$, s$, age
   LET  S$ =ucase$(s$)
   PRINT "------------------------------"
   IF (s$ ="F" and age<20) or(s $ ="M" and age<22) then
       PRINT  n$, "lawless age!"               ! 非法年龄
   END IF
   IF (s$ ="F" and age>=20)or(s$ ="M" and age>=22) then
       PRINT  n$, "lawful age."                ! 合法年龄
   END  IF
   PRINT "----------------------------------------"
   INPUT  prompt "Continue(y/n)?": pS
   IF  ucaseS(p$)="N" then
```

```
        EXIT  DO
    END  IF
LOOP
END
```

```
--------------------------------------------
Ok. RUN
Name, Sex("f" or "m"), Age=? ZhangSan, m, 21
-----------------------------------
Zhang San                    lawless age
-----------------------------------
Continue(y/n)? n
Ok.
```

5.1.3 接收一整行字符的 INPUT 语句

为了适应文字处理的需要，True BASIC 的 INPUT 语句还允许接收一整行字符(其中可含逗号，双引号不需双写)，作为一个字符串，语句格式为:

LINE　INPUT　x$

功能：当 LINE　INPUT　x$ 语句被执行时，屏幕上显示问号“?”等待输入一行字符(以回车作为输入结束)。

例 3　输入一组书名，统计其中共有多少本书的书名中出现“Computer”一词。当输入内容为某特特殊行(如“9999”)时，程序自动结束。

程序:

```
LET  i=0
DO
    LINE  INPUT  prompt "Enter name of book: ":  BookName$
    IF   BookName$ ="9999" then                 !如果为"9999"则退出循环
         EXIT DO
    END  IF
    IF  pos(ucase$( BookName$ ), "COMPUTER")>0 then
        LET  i=i+1
    END  IF
LOOP
PRINT  "TOTAL=";  i
END
```

5.2　PRINT　语句

PRINT 语句是 True BASIC 程序输出数据的主要语句. 它的形式多样，用法灵活，技术巧妙。

5.2.1 PRINT 语句格式

向屏幕输出表达式之值的语句格式:

PRINT　expr1，expr2，…，exprN

功能：将 PRINT 后面所列的表达式 expr1，expr2，…，exprN 的值显示在屏幕上。

其中，表达式 expr1，expr2，…，exprN 既可以是数值表达式，也可以是字符串表达式。

在表达式之间既可用逗号“,”分隔，也可用分号“;”分隔。执行的效果是分别使表达式的值以“标准格式”和“紧凑格式”显示在屏幕上。

True BASIC 将屏幕窗口的 80 列(屏幕上每行有 80 个字符位置)分为 5 个标准区，每个标准区占 16 个字符宽。由逗号分隔的每个表达式占 1 个标准区。5 个区被占满则自动向下移动。如果表达式之间是分号，则数字(包括符号位)之间以 1 个空格分隔(但字符串之间不作分隔)。

如果 PRINT 语句中的最后一个表达式 exprN 的后面带有逗号或分号，则会以标准格式或者紧凑格式影响下一个 PRINT 语句的输出。如果关键字 PRINT 后没有任何表达式或符号，则该语句起换行的作用。

5.2.2　带 TAB(C)和 TAB(R，C)的 PRINT 语句

语句格式：

(1)　PRINT　TAB(C); expr

(2)　PRINT　TAB(R，C); expr

其中，TAB(C)和 TAB(R，C)分别为指定屏幕上列位置(C 列)和行列位置(R 行 C 列)的定位函数。

语句(1)的功能：将表达式的值显示在屏幕的第 C 列处。

语句(2)的功能：将表达式的值显示在屏幕的第 R 行第 C 列处。

例 4　利用 TAB(C)函数显示下面的图形：

```
      *
    * * *
  * * * * *
* * * * * * *
```

程序：

```
LET  m=4
LET  i=1
DO
     PRINT  tab(41-i*2);             ! 将显示位置移至 41-2i 列
     LET  n=2*i-1                    ! 第 i 行″*″的个数
     PRINT  repeat$(″* ″, n)         ! 输出 n 个″*″
     LET  i=i+1
     IF  i>m then
         EXIT  DO
     END  IF
LOOP
END
```

例 5　利用 TAB(R，C)函数显示 $y=\mathrm{sech}(x)=2/(e^x+e^{-x})$的函数图象。

程序：

```
PRINT  tab(21, 1); repeat$("_", 80)              ! 显示 X 轴
FOR  y=1 to 25
     PRINT  tab(y, 40); "|"                       ! 显示 Y 轴
NEXT  y
FOR  x=-5 to 5 step 0.4                           ! 显示函数图象
     LET  y=2/(exp(x)+exp(-x))
     PRINT  tab(20-15*y, 7*x+40); "*"
NEXT x
END
```

当程序中使用了 TAB(x，y)函数，程序执行时屏幕会自动转到全屏幕显示输出结果状态(称为屏幕的正文模式)，显示结果后即停留在此状态。当按任一键后，屏幕返回分上下窗口的基本状态。

5.2.3　带 USING “格式描述字符串”的 PRINT 语句

语句一般格式：

PRINT　USING “格式描述字符串”: expr

功能：使表达式 expr 按照“格式描述字符串”所描述的形式输出。

最常用的格式描述字符有：

空格——表示该位置为空；＃——表示该位置为数字；%——表示该位置为数字，无数字时补零；·——表示数值的小数点位置；，——表示逗号的位置；^^^^——表示数值的指数部分的位置。

例 6　利用带 USING 描述的 PRINT 语句输出 X，X^2，$\sqrt{X}$及 $X^2\log_{10}X$ 的数值表(自变量 X 从 10 变到 100，间隔 10)。

程序：

```
LET  x=10
LET  i=1
PRINT  tab(2, 4); "X"; tab(2, 17); "X*X";
PRINT  tab(2, 31); "SQR(X)"; tab(2, 50); "X*X*LOG10(X)"
DO
    PRINT  tab(3+i, 1);
    PRINT  using"%%%": x;
    PRINT  tab(3+i, 10);
    PRINT  using"##########": x*x;
    PRINT  tab(3+i, 30);
    PRINT  using"##.##########": sqr(x);
    PRINT  tab(3+i, 50);
    PRINT  using"#.######^^^^": x*x*log10(x)
    LET  x=x+10
    IF  x>100 then
```

```
        EXIT  DO
      END  IF
      LET  i=i+1
LOOP
END
-----------------------------------------------------
Ok. RUN
  X          X * X          SQR(X)            X * X * LOG10(X)
  10          100          3.1622776602          1.000000e+02
  20          400          4.4721359550          5.204120e+02
  30          900          5.4772255751          1.329409e+03
  40         1600          6.3245553203          2.563296e+03
  50         2500          7.0710678119          4.247425e+03
  60         3600          7.7459666924          6.401345e+03
  70         4900          8.3666002653          9.040980e+03
  80         6400          8.9442719100          1.217978e+04
  90         8100          9.4868329805          1.582936e+04
 100        10000         10.0000000000          2.000000e+04
```

5.2.4 向打印机输出

True BASIC 设置了使用打印机作输出的程序语句和命令。

(1) 打印当前内存中的程序清单　当确认打印机已联机后，在命令窗口的 Ok.之右边键入 LIST 命令，即可在打印机上打印出程序清单。

(2) 打印屏幕上确认正确的计算处理结果　对程序需作如下处理：

a) 在程序中的适当位置(一般是程序开头处)，使用打开通道号并连接打印机的语句

OPEN　#expr:printer

b) 将程序中所有的 PRINT 语句改为向指定通道输出的语句

PRINT　#expr:

c) 在程序中适当位置(一般是程序尾部，END 语句之前)，使用关闭通道的语句

CLOSE　#expr

运行经过上述修改的程序，即可在打印机上输出程序运行结果。

上述 3 个语句中的#expr，称为程序与打印机相连接的通道号(或者称为文件号)。它由“#”号后跟一数值表达式构成，例如：#1，#2，#3，#102 等，expr 可取 1～9999 的整数(但 True BASIC 只允许同时打开 10 个通道)。printer 是特指打印机。

注意：在打印机上 TAB(C)函数仍然有效，但不能使用 TAB(CR，C)函数

例 7　计算 0～π(间隔 10°)的正弦、余弦和正切函数值并输出到打印机。

程序：

```
OPEN  #3:printer                    ! 打开 3 号通道并与打印机相连
OPTION ANGLE  degrees               ! 三角函数选择以角度为单位
LET  x=0
PRINT  "x", "sinx", "cosx", "tgx"
PRINT  #3: "x", "sinx", "cosx", "tgx"        ! 向 3 号通道输出
DO
    LET  y1=sin(x)
    LET  y2=cos(x)
    LET  y3=tan(x)
    PRINT  x, y1, y2, y3
    PRINT  #3: x, y1, y2, y3                 ! 向 3 号通道输出
    LET  x=x+10
    IF  x>180 then
        EXIT  DO
    END  IF
LOOP
CLOSE  #3                                    ! 关闭第 3 号通道
END
```

5.3 READ/DATA 语句与 RESTORE 语句

程序运行所需要的数据，除了可以由外部设备(例如键盘)提供之外，在程序内部也可以由 DATA 语句包含数据，作为程序的一部分提供给程序中的变量使用。这些数据随同程序装入内存时，它们被有序地存放在内存的数据区中。

5.3.1 DATA 语句

在程序中用一个或多个 DATA 语句将数据置入内存数据区。语句格式:

DATA item1, item2, …, itemN

其中，数据项 item1, item2, …, itemN 既可以是数字常量，也可以是字符串常量。当程序运行时，所有 DATA 语句后面所带的数据，将按 DATA 语句出现的次序及数据项排列的前后，依次存入内存数据区。

5.3.2 READ 语句

使用 READ 语句，可从内存数据区中读出数据并传递给程序变量。语句格式:

READ Var1, Var2, …, VarN

其中，变量 Var1, Var2, …, VarN 既可以是数字变量，也可以是字符串变量。若为数字变量，传递给它的数据项就必须是数字；若为字符串变量，传递给它的数据项就必须是字符串，否则都会出错。另外，如果 READ 语句读数时，内存数据区中的数据项数不够，True BASIC 也会给出出错信息(Reading past end of data，参见附录 6)。

例 8 用 Read/Data 语句改写例 1 的程序。

```
DATA 18, 15, 27
READ  a, b, c
LET  p=(a+b+c)/2
LET  s=sqr(p*(p-a)*(p-b)*(p-c))
PRINT  "a="; a, "b="; b, "c="; c
PRINT  tab(16); "S="; s
END
```

例 9　按下面的公式计算 R 的值。

$$R=\sqrt[4]{\frac{PA^2C}{4\pi M^2KD}\epsilon^{-2.3L}}$$

程序:

```
DATA  2.5e5, 1.7, 50, 0.1, 2e-12, 9.12, 1
READ  p, a, c, m, k, d, l                    ! 读入这组参数的数值
LET  f1=p*a*a*c*exp(-2.3*l)
LET  f2=4*pi*m*m*k*d
LET  r=(f1/f2)^(1/4)
PRINT  "r="; r
END
---------------------------
Ok. RUN
r=35457.4
Ok.
```

如将程序中用 READ 语句为 7 个变量读数改用 INPUT 语句实现时，就需要从键盘输入 7 个参数数值。当需要计算多组参数时，每次键入多个变化较大的参数，不仅影响运算的速度，而且又比较麻烦、易错。对这类需要输入成组参数的问题，把参数值置入 DATA 语句中，由 READ 语句自动读取是更有效的数据输入办法。

当程序开始执行时，标示内存数据区数据的“数据指针”，总是指向第 1 个 DATA 语句的第 1 个数据。每个 READ 语句按变量顺序，依次读取由该程序所有 DATA 语句置入的数据项。每读取一项数据,“数据指针 ”就顺次指向下一项数据。一般，数据指针所扫过的数据，就不能再读取。如果需要重新读取，要用 RESTORE 语句，将“数据指针”恢复到数据区的第 1 项数据处。这样，该数据区的数据将又可重新被读取。

5.3.3　RESTORE 语句

用 RESTORE 语句将数据指针恢复到数据区第 1 个数据处。语句格式:

RESTORE

当这个语句被执行时，将会把处于数据区任何位置上的数据指针拨回到第 1 个数据处，从而能重新使用已经读过的数据，这对重复使用大批量数据是十分有用的。

例 10　按公式计算一组观测数据 x_i(i=1, 2, …, n) 的标准差

$$\mathrm{D}=\sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(x_i-\bar{x})^2}$$

其中，n 为数据的个数，$\bar{x}$ 为 n 个数据的算术平均值。

程序:

```
DATA  4, 5, 17, 12, 15, 11, 2, 7, 21, 14
LET  n=10
LET  s=0
LET  i=1
DO
```

```
        READ  x                        ! 读取 DATA 中的数据
        LET  s=s+x
        LET  i=i+1
        IF  i>n  then
            EXIT  DO
        END  IF
    LOOP
    LET  a=s/n
    RESTORE                            ! 恢复数据指针
    LET  m=0
    LET  j=1
    DO
        READ  x                        ! 再次重头读取 DATA 中的数据
        LET  m=m+(x-a)*(x-a)
        LET  j=j+1
        IF  j>n  then
            EXIT  DO
        END  IF
    LOOP
    LET  d=sqr(m/(n-1))
    PRINT  "D="; d
    END
    -----------------------
    Ok. RUN
    D=5.86174
    Ok.
```

例 11　已知 x 分别为 45°，50°，55°，…，85°，90°，计算下式的值.

$$g=\frac{\pi}{2}+\frac{(\sin x)^3+(\cos x)^2}{2}+\frac{1}{3(\sin x)^3}$$

程序:

```
    OPTION  ANGLE degrees
    DO
        READ  x
        IF  x=9999 then
            EXIT  DO
        END  IF
        LET  s=sin(x)
        LET  s=s*s*s
        LET  c=cos(x)
        LET  c=c*c
        LET  g=pi/2+(s+c)/2+1/(3*s)
    PRINT  "x=";  x,  "g=";  g
LOOP
DATA  45, 50, 55, 60, 65, 70, 75, 80, 85, 90, 9999
END
```

例 12　打印一张电话表。

程序:

```
! phone  list
PRINT  "Name"; tab(30); "Number"
PRINT
DO
    IF   end data then)   EXIT DO                ! 如果数据结束则退出循环
    READ   n$ , p$
    PRINT   n$ ; tab(30); p$
LOOP
DATA  Zhang San, 111, Li Shi, 205, Wang Wu, 5621-323
END                                              !program end.
```

在程序的 DATA 中电话号码 111，205，5621-323 是以字符串形式提供给 p $ 的。

习　题　5

(1) 输入／输出在程序中有什么意义?

(2) Ture BASIC 有几种输入／输出语句? 说明其格式及格式中各个部分的作用。

(3) 试求如下情况的三角形面积 S:

①输入底边 a，高度 h 时，$S=\mathrm{ah}/2$

②输入两角(单位: 弧度)t_1、t_2 及其所夹边 a 时，$S=\dfrac{a^2\sin t_1\sin t_2}{2\sin(t_1+t_2)}$

(4) 试求底面半径 $r=5$，高 $h=12$ 的园锥体的体积 $V=\pi r^2h/2$。

(5) 给出华氏温度 F，试求摄氏温度 C 和绝对温度 Z。

提示:　$C=5/9(F-32)$，　$Z=C+273.16$

(6) 鸡兔同笼问题: 已知鸡兔总头数 H，总脚数 F，求鸡兔各几只?

(7) 输入复数 A，B，计算下式:

$$Z_1=A+B;\quad Z_2=A-B;$$
$$Z_3=A\times B;\quad Z_4=A/B.$$

(8) 在△ABC 中: $AC=r_1$，$BC=r_2$ 以及$\angle ABC=\alpha$，求 AB:

$$AB=\sqrt{r_1^2+r_2^2-2r_1r_2\cos\alpha}$$

(9) 打印 6 位正弦函数表: sin(x)。要求: $x\in[0°, 360°)$，$\Delta x=1$

(10) 做一个数字钟，在屏幕中心以 hh: mm: ss 格式显示时间。

(11) 统计下列20个数的算术平均数:

0.1，2.3，2.7，0.9，2.13，5.34，5.1，6.2，2.23，6.4，1.15，4.43， 3.21，0.5，4.88，3.34，1.65，6.91，0.02，3.28

(12) 用字符“＊”画出函数 $y=x^2$ 的图象。

(13) 按如下格式打印杨辉三角形(共印 10 份)。

```
        1
      1   1
    1   2   1
  1   3   3   1
```

提示:

利用二项式定理可知第 i 行，第 j 列的数为

$$C_{i-1}^{j-1}=\frac{(i-1)!}{(i-j)!(j-1)!}$$

(14) 打印算术中的九九乘法表。

第6章　判断结构

在处理实际问题过程中，需要根据不同的情况，选择不同的处理方案，这就是所谓“逻辑判断”的过程。对于这种过程，仅用顺序执行一次的语句结构是无法表达的。在各种算法设计中会频繁出现逻辑判断问题，因此，程序设计中必须引入逻辑判断机制。True BASIC 中运用判断结构和逻辑函数，可解决各种判断问题。

判断结构是根据给定的条件(包括由各种简单条件组合而成的复合条件)，对程序的语句序列作有选择的执行，使语句的执行顺序和次数都可以灵活地变化。

6.1　算法及其逻辑框图表示

6.1.1　算法的概念

在计算机科学中，算法是指计算解题的精确有效的方法步骤，能被机械执行的动作(或称为规则、指令)序列。这里所指的算法是用语言描述的问题的求解“程序”，而计算机的程序则是用计算机语言所表达的算法。

能够用算法来求解的函数或问题，称为可计算函数或可计算问题。一般，一个算法有下列 5 个特征:

(1) 输　入——一个算法有零个或多个输入；

(2) 确定性——算法的每一步骤必须有完全确定的意义，不可有二义性；

(3) 有穷性——一个算法必须在执行有穷步骤之后结束；

(4) 输　出——一个算法有一个或多个输出；

(5) 可行性——算法中的每一基本操作必须能有效地执行，最后应得到确定的结果。若算法中存在不可执行的操作，整个算法就是无效的。

算法与通常的计算过程密切相关，但在有穷性上通常的计算过程并不一定作出要求。因此，计算过程不一定终止。这一点正是算法与数学中的“计算方法”的区别。

6.1.2　算法的逻辑框图表示

算法除了用自然语言和程序描述之外，在程序设计中也常用直观、形象、逻辑关系明确的流程框图来表述。

流程框图用不同的几何图框表示各种程序操作的类型，并且在框内标注简明的文字、符号，表示某些处理的内容，用流线箭头标示操作的逻辑次序。图 6-1 是一些常用的流程框图符号。

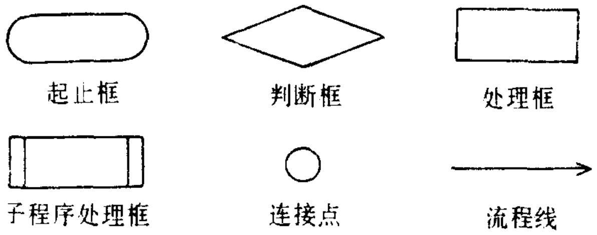


图 6-1　常用程序流程框图符号

例 1　用流程框图表示下列累加求和过程:

$$1+2+3+\cdots+k+\cdots+n=\sum_{i=1}^{n} i$$

设置变量 S 和变量 i 分别记录和数及求和项的值，并将它们的初值置为零。

这一有规律地重复求和运算过程，用逻辑框图表示如图 6-2。除了开始、终结 2 个框

外，其他 6 个框分别表示：

(1) 输入求和的总项数 n；

(2) 对变量 S、i 置初值；

(3) 将 i 加 1 后，再赋给 i；

(4) 作求和运算：S 加 1，再赋给 S；

(5) 判断 i 是否等于 n，如果相等则做(6)，否则返回(3)；

(6) 输出 S，结束求和过程：

例 2　著名的辗转相除(又称为欧几里德算法)求 2 个正整数 *m*，*n* 的最大公约数的算法，分求余、判断和置换 3 步辗转进行，其逻辑流程如图 6-3。除了开始、结束 2 个框外，其他 5 个框分别表示：

(1) 输入正整数 m，n；

(2) 作求余运算：r = mod(m，n)；

(3) 判断 r 是否为零？为零则做(5)，否则做(4)；

(4) 做置换：将 n 赋给 m，r 赋给 n，返回(2)；

(5) 输出最大公约数。

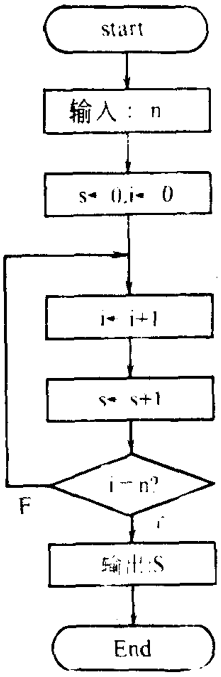

start
输入：n
s← 0,i← 0
i← i+1
s← s+1
i = n?
F
T
输出:S
End

图 6-2　累加求和运算逻辑框图

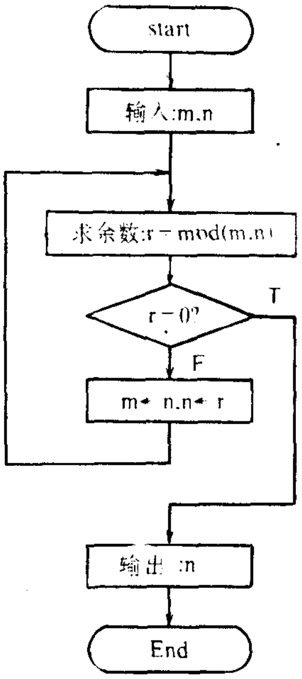

start
输入:m,n
求余数:r = mod(m,n)
r = 0?
T
F
m← n,n← r
输出 :n
End

图 6-3　辗转相除求最大公约数的逻辑框图

6.2　IF 型判断结构

True BASIC 中以 IF 关键字开头的一系列逻辑判断结构和语句，可方便地用来表达算法中经常出现的分支处理、条件选择、真伪判别等问题。

6.2.1　基本的判断结构

结构的一般格式：

```
IF <c> THEN
    ……
    ……  } (语句块)
    ……
END IF
```

其中，<c>表示判断的条件，语句块表示一组语句构成的序列。

功能：以 IF / END IF 为头 / 尾的判断结构被执行时，首先判断条件<c>是否成立？如果条件成立则执行语句块，然后做 END IF 后面的语句(或结构)；若条件不成立时，则不执行语句块，而是直接做 END IF 后的语句(或结构)。

其执行过程可由图 6-4 逻辑框图表示。其中，T(rue)表示条件成立，F(alse)表示条件不成立。

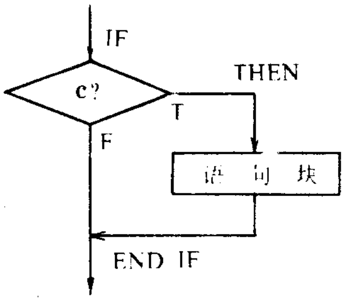


图 6-4　IF / END IF 型判断结构逻辑框图

第 5 章的例 1，根据三角形 3 条边 *a*，*b*，*c* 的值计算三角形面积 *S* 的值，但当输入的 *a*，*b*，*c* 其值不能构成三角形时，则会在计算 *S* 时发生错误。因此，程序应该在 INPUT a，b，c 语句后加上判断结构，使程序避免出错。程序可改为：

```
INPUT a, b, c
LET  p=(a+b+c)/2
LET  t=p*(p-a)*(p-b)*(p-c)
IF  t<=0 then
STOP                          ! 程序中断语句
END  IF
LET  s=sqr(t)
PRINT  s
END
```

6.2.2　简单的 IF 语句

如果在基本判断结构的语句块中仅仅包含 1 个语句，如上例所示，对于这种基本判断结构的特例，可以用简单 IF 语句来表达。语句格式：

IF <c> THEN <s>

其中，<c>为判断条件，<s>为 1 个语句。则上面的程序可改写为如下等效程序：

```
INPUT  a, b, c
LET  p=(a+b+c)/2
LET  t=p*(p-a)*(p-b)*(p-c)
IF  t<=0 then  STOP
LET  s=sqr(t)
PRINT  s
END
```

6.2.3　带 ELSE 的判断结构

一般格式：

```
IF <c> THEN
    ......
    ......  } (语句块1)
    ......
ELSE
    ......
    ......  } (语句块2)
    ......
END IF
```

功能：由 IF／ELSE／END IF 组成的判断结构被执行时，首先判断条件<c>是否成立，如果条件为真(True)，则执行语句块 1，并且结束判断过程，执行 END IF 后的语句；否则(即条件为假(False))，就执行语句块 2，然后接着执行 END IF 以后的语句。

结构的逻辑流程如图 6-5 所示。

例 3 寻找变量 *a*，*b*，*c* 中的最大值。其判断过程由图 6-6 给出。

根据框图写出对应的求解程序(不使用标准函数 max(x，y))：

```
INPUT a, b, c
IF   a>=b  then
     IF   a>=c  then
          LET  z=a
     ELSE
          LET  z=c
     END  IF
ELSE
     IF   b>=c  then
          LET  z=b
     ELSE
          LET  z=c
     END  IF
END  IF
PRINT  z
END
```

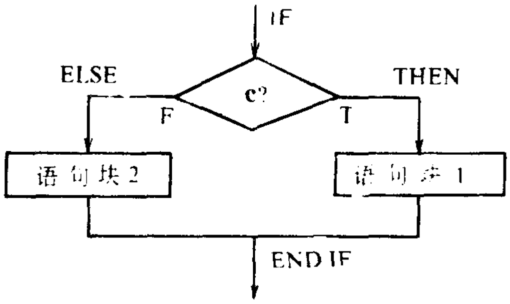


图 6-5 IF／ELSE／END IF 型判断结构逻辑框图

在程序中 3 次(外层 1 次，内层 2 次)使用了带 ELSE 的判断结构。

6.2.4 简单的带 ELSE 的 IF 语句

对于语句块 1 及语句块 2 都是 1 个语句的情形，可以由简单的带 ELSE 的 IF 语句来表达。语句格式：

IF <c> THEN <s_1> ELSE <s_2>

其中，<s_1>和<s_2>各为 1 个语句。

因此，例 3 的程序可简化为：

```
INPUT  a, b, c
IF  a>=b  then
    IF  a>=c  then  LET  z=a  else  LET  z=c
ELSE
    IF  b>=c  then  LET  z=b  else  LET  z=c
END  IF
PRINT  z
END
```

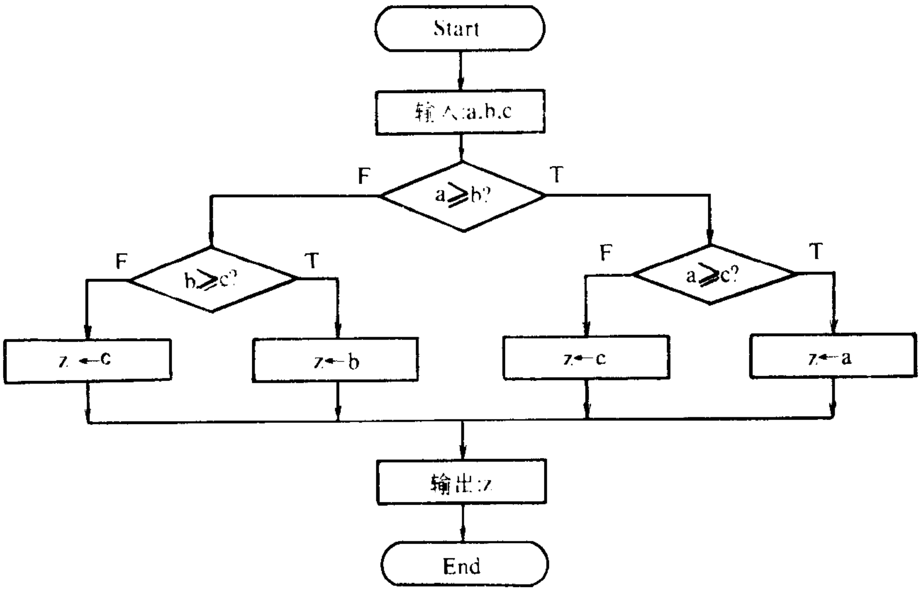


图 6-6　寻找 a，b，c 中最大值过程逻辑框图

6.2.5　扩展的带 ELSE IF <c> THEN 的判断结构

True BASIC 为判断更复杂的问题，提供了带 ELSE　IF 的判断结构。一般格式:

```
IF <c1> THEN
   ……
   ……} (语句块1)
   ……
ELSE IF <c2> THEN
   ……
   ……} (语句块2)
   ……
ELSE IF <c3> THEN
   ·
   ·
   ·
ELSE IF <ci> THEN
   ……
   ……} (语句块i)
   ……
ELSE IF <ci+1> THEN
   ·
   ·
   ·
ELSE
   ……
   ……} (语句块)
   ……
END IF
```

其中的 ELSE IF $<c_i>$ THEN 及语句块 i 可以使用任意多个(或零个)。结构的逻辑框图如图 6-7 所示。

程序按次序对从条件 $<c_i>$ 开始的一系列条件作检测判定。当条件 $<c_1>$ 为真时，则执行语句块 1，然后结束整个判断；当条件 $<c_1>$ 为假时，就顺次检测条件 $<c_2>$，为真则执行语句块 2，结束整个判断。依此类推，逐次检验所列条件。如果所列条件都不成立，则执行 ELSE 后的语句块，然后结束整个判断过程。

例 4　对形如 $ax^2+bx+c=0$ 的二次方程求根。当根的判别式 (b^2-4ac) 为正时，方程有两实根；为零时有一实根；为负时有两复根。用带 ELSE IF 的判断结构求解。

程序:

```
INPUT  prompt "a, b, c=": a, b, c
IF  a=0 and b=0 then      ! 如果 a=0 且 b=0 则输出
    PRINT "No meaning" ! 方程无"意义"字样
ELSE  IF  a=0  then
        LET  x=-c/b
        PRINT  "x="; x
```

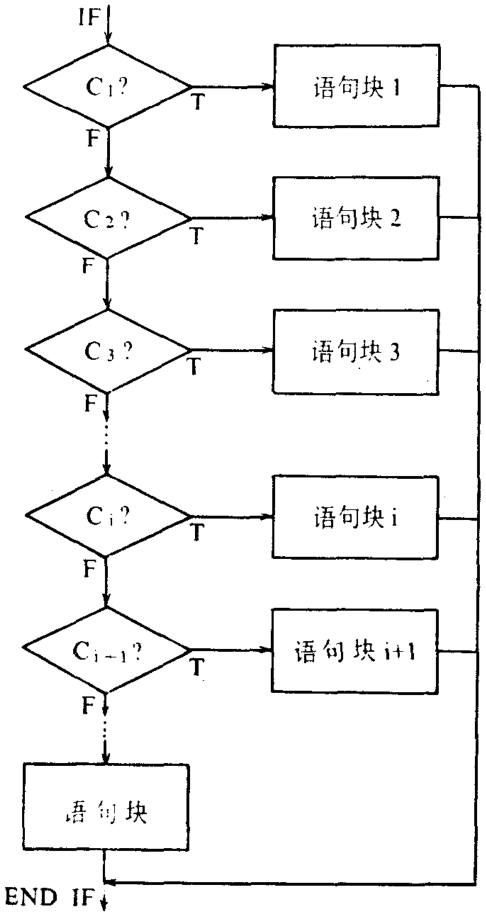


图 6-7　扩展型带 ELSE IF <c> THEN 判断结构逻辑框图

```
         ELSE
          LET  d=b*b-4*a*c
         IF  d>=0 then
              LET  r=sqr(d)
              LET  x1=(-b+r)/(2*a)
              LET  x2=(-b-r)/(2*a)
              PRINT  "real roots: ";
              PRINT  "x1="; x1, "x2="; x2
         ELSE
              LET  r=sqr(-d)
              LET  r1=-b/(2*a)
              LET  i1=r/(2*a)
              PRINT  "Complex roots: ";
              PRINT  "x1="; r1; "+i*"; i1,
              PRINT  "x2="; r1; "-i*"; i1
         END  IF
    END  IF
    END
```

例 5　将学生考试成绩分 a，b，c，d 档进行统计。a 档为 $90\leqslant x\leqslant 100$，b 档为 $75\leqslant x<90$，c 档为 $60\leqslant x<75$，d 档 $x<60$。

程序:

```
    DO
         IF  end data then   EXIT DO
         !如果 data 中数据已完结，则退出循环。
         READ  x
         IF  x<=100 and x>=90 then
              LET  a=a+1
         ELSE IF  x<90 and x>=75 then
              LET  b=b+1
         ELSE IF  x<75 and x>=60 then
              LET  c=c+1
         ELSE
              LET  d=d+1
         END  IF
    LOOP
    PRINT  "a=";a,"b=";b,"c=";c,"d=";d
    DATA  75,88,63,44,95,90,58,68,99
    END
```

6.3　SELECT CASE 判断结构

SELECT CASE 形式的判断结构对多种情况进行检测判断，它顺次地对 CASE 后面所列出的测试情况 tests 进行匹配。其功能类似于 IF／ELSE IF 结构。一般格式:

```
SELECT  CASE expr
        CASE  test1, test2, …
              (语句块 1)
        CASE  test3, test4, …
```

```
                (语句块 2)
            CASE   testi, …
                (语句块 i)
               •
               •
               •
            CASE   ELSE
                (语句块)
      END SELECT
```

其中，expr 是数字(或字符串)表达式(也称为选择表达式或情况下标)。

True BSAIC 在执行 SELECT CASE expr 语句时，先计算出表达式 expr 的值，然后按次序对 testi 作检测。当测到一个与 expr 值相匹配的 testi 就执行其下的语句块，接着结束选择，执行 END SELECT 之后的语句。如果找不到相匹配的 testi 就执行 CASE ELSE 后的语句块，然后结束检测。需注意，不要使选择表达式会同时选中执行两个不同的语句块。

例 6 投掷两个骰子(为均匀六面体，每面上有 1～6 个数字之一。这里用产生 1～6 的随机整数代替掷骰子)的游戏。投掷规定，如果开始掷得的点数为 2 或 3 或 12，则计输(Lose)；掷出的点数为 7 或 11 则计赢(Win)。若掷出的点数既不是 2，3，12，又不是 7，11 则不计输赢，允许继续再掷。要是再掷得点数为 7，那么这一次就算输了；要是再掷出得点数是一个与第 1 次掷出得点相同的数，这一次就算赢了。游戏总共玩 10 次，每次显示出输赢情况。

程序：

```
RANDOMIZE
LET  g=1
DO
   LET  d1=int(6*rnd+1)
   LET  d2=int(6*rnd+1)
   LET  d=d1+d2
   PRINT  d;
   SELECT  CASE  d
   CASE 2, 3, 12
      PRINT  ″Loss″
   CASE 7, 11
      PRINT  ″Win″
   CASE  ELSE
        LET  p=d
        DO
           LET  d1=int(6*rnd+1)
           LET  d2=int(6*rnd+1)
           LET  d=d1+d2
           PRINT  d;
           IF  d=7 or d=p then  EXIT DO
        LOOP
        IF  d=p then  PRINT "Win" else  PRINT  "Loss"
    END  SELECT
    LET  g=g+1
    IF  g>10 then EXIT DO
LOOP
END
```

程序中的 d(每次掷 2 颗骰产生的点数)即为 expr 的值(情况下标)。

从 SELECT CASE expr 语句起至 END SELECT 语句止的结构中，CASE 2，3，12 及 CASE 7，11 分别规定了 2 种检测情况.当 d 为 2，3 或 12 时，PRINT "Lose"被执行；当 d 为 7 或 11 时 PRINT "Win"被执行；当 d 不属于上述 2 种情况则 LET p=d 语句至 IF d=p THEN PRINT "Win" ELSE PRINT "Lose"语句之间的语句块被执行。

对于情况的检测除了数值常数外，还可以是对定界内的常数或字符串或一个关系式的检测。

(1) test 为定界内的数值常数。例如:

CASE 1 to 10；CASE −1 to 11；CASE 1949 to 1990.

(2) test 为定界内的字符串常量。例如:

CASE I to N；CASE A to F；CASE W to Z.

(3) test 为情况枚举。例如:

CASE "Age", "sex", "M", "W"；CASE 2，5，8，11

(4) test 为关系说明。例如:

CASE is>100；CASE is< >50；CASE is< =0.

(5) test 为(1)，(2)，(3)，(4)的组合。例如:

CASE −5 to 5，9，is>13；CASE x to z, "A", "B", "C"。

虽然，SELECT CASE 结构中，CASE ELSE 部分是可选择的，它下面的语句块仅当 CASE tests 的所有检测均无效才被执行。但是，在应用 SELECT CASE 结构时，写出 CASE ELSE 部分对保障程序安全是有益的。因为当情况表达式的值在所有 CASE tests 中都检测不到相匹配的项，且结构中又无 CASE ELSE 部分时，程序运行将出现错误(NO CASE selected，but no CASE ELSE，参见附录 6)，并停止运行。

6.4 程序举例

例 7 计算并输出区间[0，6]上分段函数

$$y=\begin{cases} x-4.5, & (0\leqslant x\leqslant 2)\\ e^{x}+e^{2x}, & (2<x\leqslant 4)\\ x^{2}-2x+1, & (4<x\leqslant 6)\end{cases}$$

的函数值。

程序:

```
INPUT  x
IF  x<0 or x>6 then
    PRINT "x doesn't belong to [0, 6]"
ELSE
```

```
    IF  x>=0 and x<=2 then  PRINT  x-4.5
    IF  x>2 and x<=4 then  PRINT  exp(x)+exp(2*x)
    IF  x>4 and x<=6 then  PRINT  x^2-2*x+1
END IF
END
```

例 8 用二分法求方程 $f(x)=0$ 在区间(a,b)上的一个近似实根。其中，$f(x)$为在区间上连续且 $f(a)$和 $f(b)$异号的函数，求方程

$$x^3-2x-5=0$$

在(2，3)内的近似根。

程序：

```
INPUT  a, b, e                        ! e为求近似实根的允许误差
LET  c=(a+b)/2
DO
    LET  fc=c^3-2*c-5
    IF  fc<e then EXIT DO
    LET  fa=a^3-2*a-5
    IF  fc*fa<0 then
        LET  b=c
    ELSE
        LET  a=c
    END IF
    LET  c=(a+b)/2
LOOP
PRINT "x="; c
END
```

例 9 猜字母游戏。计算机随机设定一个英文字母，游戏者不断地从键盘输入所猜的字母，直到猜对为止。当猜测相差太远时计算机给出信息“differ greatly”(差很远)；如果逐步接近时则分别给出“differ”(不对)，“near”(接近)，“almost”(差一点)的信息。猜对之后，程序还会询问是否继续做下一次游戏，要求回答(Y／N?)。

程序：

```
RANDOMIZE
LET  r=int(26*rnd+65)        ! 随机产生英文字母的ASCII码(65~90)。
DO
    INPUT  prompt"enter a letter: ": x$
    IF  ord(x$)>90 then  LET  s=ord(x$)-32  else  LET  s=ord(x$)
    LET  c=abs(s-r)
    SELECT  CASE c
    CASE  is>20
            PRINT "differ greatly"
    CASE  10 to 20
            PRINT "differ"
    CASE  5 to 9
          PRINT  ″near″
    CASE  1 to 4
          PRINT  ″almost″
    CASE  0
          PRINT  ″right″
          PRINT  ″guess again(y / n)?″
          INPUT   g $
          IF   g $ = ″n″ or g S = ″N″ then   EXIT DO
          LET   r = int(26 * rnd+65)
    CASE  ELSE
          PRINT  ″Please input a letter. ″
    END  SELECT
LOOP
END
```

例 10 设计一个整数算术四则运算练习程序。程序分别提供做加法、减法、乘法、除法和退出程序 5 种选择，并询问每次练习几位数的运算和每次做几题，每做完一次给出得分。

程序:

```
DO
    ! display   menu. 显示菜单
    PRINT  ″1. Addition.″                                  ! 1 表示加法
    PRINT  ″2. Subtraction.″                               ! 2 表示减法
    PRINT  ″3. Multiplication.″                            ! 3 表示乘法
    PRINT  ″4. Division.″                                  ! 4 表示除法
    PRINT  ″5. Exit.″                                      ! 5 表示退出
    ! select + ,  -,   * ,   /  or exit
    INPUT prompt ″select 1--5?″:  c                        ! 选择加，减，乘，除或退出
    IF   c< =0 or c> = 5 then   EXIT DO
    INPUT   prompt ″How many places?″:  w                  ! 回答做几位数运算
    INPUT   prompt ″How many questions?″:  n               ! 回答做几道题
    RANDOMIZE
    LET   s=0
    LET   i=1
    DO
        DO
            LET   a=int(10^w * rnd+1)
            LET   b=int(10^w * rnd+1)
            IF   a>b then EXIT DO
        LOOP
        SELECT   CASE c
        CASE  1
            LET  p $ =″+″
            LET  q=a+b
        CASE 2
            LET  p $ =″-″
            LET  q=a-b
        CASE 3
            LET  p $ =″x″
            LET  q=a * b
        CASE 4
            LET  p $ =″/″
            LET  q=int(a / b)
            LET  r=mod(a, b)
        END  SELECT
        !
        PRINT  a; p $ ; b; ″=″;
        ! 如果作除法需回答商数 q 和余数 r
        IF  c=4 then  INPUT  prompt ″? ... q, r=″: u, v  else  INPUT  u
        IF  c=4 then
            IF  u=q and v=r then  LET  f=1 else  LET  f=0
        ELSE
            IF  u=q  then  LET  f=1 else  LET  f=0
        END  IF
        IF  f=1 then
            PRINT ″Correct.″
            LET  s=s+1
        ELSE
            PRINT ″Wrong! Answer is″; q
            IF  c=4 then PRINT ″Remainder is″; r
        END  IF
        LET  i=i+1
        IF  i>n then  EXIT  DO
    LOOP
    PRINT  ″Your score is″; s; ″(‘; n; ’)″
    PRINT  ″Do you want another (y / n)″;
    INPUT  a $
    IF  a $ =″n″ or a $ =″N″ then  EXIT  DO
LOOP
END
```

习 题 6

(1) 执行下列语句后，变量 y 的值应当是多少?

```
①LET x = 3
  IF x > 0  then
     IF x < 4  then
     LET v = 3
     END IF
     LET y = 4
  ELSE IF x = 0 then
     LET y = 5
  ELSE
     LET y = 6
  END IF
```

```
②INPUT x
  IF x > 0 or x < 0 then
     LET y = 1
  ELSE
     LET y = 2
  END IF
  IF Y > 1 then
     LET v = x-y
  ELSE
     LET y = 2
  END IF
```

```
③LET x = 2
  SELECT CASE x
  CASE is > 4
    LET y = 1
  CASE is < 0
    LET y = 2
  CASE 1 to. 3
    LET y = 3
  CASE ELSE
    LET y = 4
  END CASE
```

(2) 编程找出一组给定数据中的最大数及最小数。

(3) 编程计算函数 $M(x)$： $M=\begin{cases} bx/H & x\leqslant a \\ a(H-x)/H & x\geqslant a \end{cases}$

(其中，a，b，H 为常数)

(4) 给出数据 i 和 x 后，试求 $f(x)$的函数值，其中 i 为 1～4 的整数。

$$f(x)=\begin{cases} x^2+1 & i=1 \\ \sqrt{x+1} & i=2 \\ 0 & i=3 \\ 1 & \\ 2 & i=4 \end{cases}$$

(5) 产生 100 个 1～100 之间的随机数，分别统计 10～20，70～90 之间的随机数的个数。

(6) 编程序统计一组数据中负偶数和正奇数的个数，并分别对负数，对正数求和。

(7) 用判断结构描述下列计算过程：

①若 $x+5>0$，则 $y=\ln(x+5)$；

②若 $r+t<10^{-6}$，则 $s_1=1$，否则 $s_2=1$；

③若 $b<a<c$，则 $x=b$ 且 $y=c$；否则 $x=y=a$。

(8) 输入 3 条线段的长度，试判断它们能构成何种三角形：

①等边三角形；　②直角等腰三角形；　③直角三角形

④等腰三角形；　⑤一般三角形；　⑥不构成三角形

第 7 章　循 环 结 构

如果某一个程序的每个语句都仅只执行一次，那么这样的程序其功能是比较局限的。而实际问题往往比较复杂，经常需要进行重复计算处理。尤其是，当每次重复的步骤相同，某些量又按照一定规律不断地变化时，如果仍以顺序语句来描述，则程序将会十分冗长。True BASIC 设有实现重复计算处理的相应语句和结构，对程序流程进行控制，简化程序。

7.1　FOR/NEXT 循环和 EXIT FOR 语句

第 2 章中介绍的 DO/LOOP 循环和 EXIT DO 语句是最基本的循环。它的结构形式为：

```
DO
      ……┐
      ……} (语句块1)
      ……┘
      IF <c> then EXIT  DO
      ……┐
      ……} (语句块2)
      ……┘
LOOP
```

相应的逻辑框图如图 7-1。

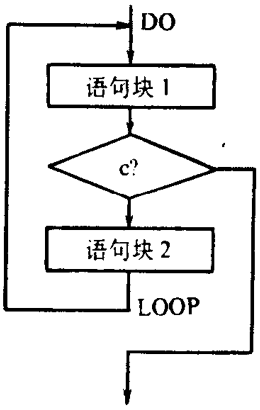


图 7-1　基本循环结构的逻辑框图

循环是否结束，由 IF 语句中的条件 <c> 控制。显然，在重复执行 DO/LOOP 之间的语句块时，必须逐步使条件<c>成立。否则，循环将无限进行下去。然而，在实际问题当中，某些类型的重复计算和处理，其重复次数是事先确定的，为了方便处理这类问题， True BASIC 设置有 FOR/NEXT 循环。

7.1.1　FOR/NEXT 循环

一般格式：

```
FOR  var=first  to  last  step  size
     ……┐
     ……}（ 循环体 ）
     ……┘
NEXT  var
```

功能：根据 first，last 和 size 有限次重复执行循环体，其中， var 是数值型变量，又称为“循环控制变量”(或“计数变量”)， first， last， size 均为数字表达式，分别表示循环的初值，终值和步长。

FOR/NEXT 循环的逻辑框图如图 7-2。

处于 FOR/NEXT 之间的语句块称为“循环体”。循环体重复执行的次数由 FOR 右边的各表达式确定。首先， var 由 first 定义。如果 var 不超过 last 则执行循环体，执行

NEXT var 时，让 var=var+size。然后返回 FOR 并再次将 var 与 last 作比较。若仍然不超过 last 就再执行循环体。如此重复，直到 var 超过 last 就退出循环，执行 NEXT 的下一语句。

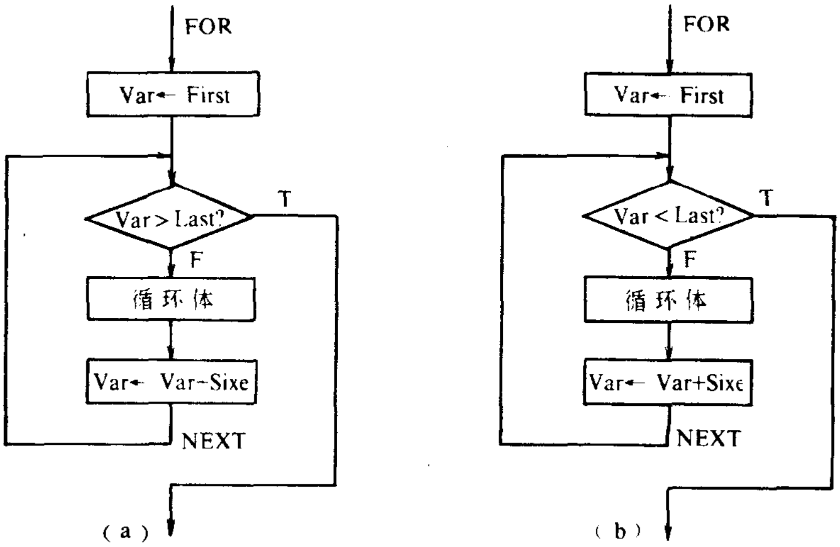

FOR
Var← First
Var > Last?
T
F
循环体
Var← Var-Sixe
NEXT
(a)
FOR
Var← First
Var < Last?
T
F
循环体
Var← Var+Sixe
NEXT
(b)

图 7-2　FOR/NEXT 型循环逻辑框图

(a)—步长为正　　(b)—步长为负

若开始执行循环时，var 就超过 last，则整个循环被跳过。在 FOR 语句中 step size 部分是可选择的。如果省略则表示 step 1。若步长为正 1，执行 NEXT 时是将循环变量的当前值加 1，即 var=var+1。

在进入 FOR 循环时，初值(first)、终值(last)及步长(size)的值是完全确定的。即便在循环体中对它们重新赋值，也不起作用，不影响循环的次数。但是，如果在循环体中，对循环变量进行赋值变换，则会改变循环的次数。因此，一般不提倡在循环体内进行改变循环变量的操作。循环正常结束时循环变量(var)的值等于：初值+步长×循环次数。

注意：若在此循环结束后继续使用循环变量，则它的这一数值已超过了终值。

例 1　统计 n 个数的算术平均值。

显然，应重复做 n 次加法求出这 n 个数之和，然后除以 n 即得平均数。

程序：

```
INPUT  prompt "n=?": n
LET  s=0
FOR  i=1 to n
     INPUT  x
     LET  s=s+x
NEXT  i
LET  m=s/n
PRINT  m
END
```

在 FOR 循环的循环体内，又可以包含 FOR 循环。这种结构形式称为循环的嵌套。内层 FOR 循环体中又可再含 FOR 循环，如此类推，构成多重循环结构。

例 2 利用两层循环，输出九九乘法表。

程序:

```
FOR  a=1 to 9
     FOR  b=1 to 9
          PRINT a;" * "; b; " = "; a * b,
     NEXT b
NEXT a
END
```

循环的具体过程为：首先，外层循环变量 a 为 1，然后进入外层循环体(FOR b / NEXT b 循环结构)执行一次产生 9 次输出(即 1 * 1 = 1，1 * 2 = 2，…，1 * 9 = 9)。然后，执行 NEXT a 语句，让 a 加 1 等于 2。返回 FOR a = 1 to 9 语句，判断 a 不大于 9，又执行循环体，再产生 9 次输出(即 2 * 1 = 2，2 * 2 = 4，…，2 * 9 = 18)。如此类推，得出九九乘法表。

例 3 输出由 1，2，3，4 这 4 个数组成的所有可能的四位数。

四位数的千，百，十，个位，分别用变量 *a*，*b*，*c*，*d* 表示。如果 *a*，*b*，*c*，*d* 都从 1 变化到 4，则组合了所有的四位数。

程序:

```
FOR  a=1 to 4
      FOR  b=1 to 4
            FOR  c=1 to 4
                  FOR  d=1 to 4
                        LET  t=a * 1000+b * 100+c * 10+d
                        PRINT  t;
                  NEXT  d
            NEXT  c
       NEXT  b
NEXT a
END
```

对于多重的循环嵌套，必须注意区别每一层循环的循环变量，并且程序应按结构将每层循环的循环体向右缩进一词写成锯齿状，以便于阅读和检查调试。

7.1.2 EXIT FOR 语句

语句格式:

EXIT FOR

功能：退出包含它的最近的一层 FOR/NEXT 循环。

在 FOR 循环的循环体中，如果含有 EXIT FOR 语句(在 FOR 循环之外不能使用此语句)，可以对循环起控制作用。当执行这个语句时即退出循环，结束这个循环过程继续执行该循环后的语句。通常 EXIT FOR 是用在条件语句中，如 IF <c> then EXIT FOR，当条件<c>成立时，就跳出该层 FOR 循环。

例 4 输入一个整数，分解出它的所有因子。

程序:

```
INPUT  n
```

```
PRINT  sgn(n);
LET  k=abs(n)
FOR  i=2 to k
     DO
          IF  mod(k, i)>0  then  EXIT  DO
          PRINT  i;
          LET  k=k/i
     LOOP
     IF  k=1 then  EXIT  FOR
NEXT  i
END
```

7.1.3 几种常用的循环程序段

在许多问题的数学处理过程中，经常遇到$\sum$(连加)和$\prod$(连乘)以及它们的组合运算，其计算过程可以用循环实现。

(1) $m=\sum_{i=1}^{n} a(i)$ 程序段:

```
!
LET  m=0
FOR  i=1 to n
     LET  m=m+a(i)
NEXT  i
!
```

(2) $m=\prod_{i=1}^{n} b(i)$ 程序段:

```
!
LET  m=1
FOR  i=1 to n
     LET  m=m*b(i)
NEXT  i
!
```

(3) $z=\sum_{i=1}^{m}\prod_{j=1}^{n} r(i,j)$程序段:

先令 $s(i)=\prod_{j=1}^{n} r(i,j)$，则$z=\sum_{i=1}^{m} s(i)$，于是有循环嵌套

```
!
LET  z=0
FOR  i=1 to m
              ┌LET s(i)=1
              │FOR j=1 to n
     求出s(i) {      LET s(i)=s(i)*r(i,j)
              └NEXT j
               LET  z=z+s(i)
NEXT  i
!
```

(4) $y=\prod_{i=1}^{m}\sum_{j=1}^{n} p(i,j)$程序段:

先令 $q(i)=\sum_{j=1}^{n} p(i,j)$，则$y=\prod_{i=1}^{m} q(i)$，于是有循环嵌套

```
!
LET  y=1
FOR  i=1 to m
                 ┌LET q(i)=0
                 │FOR j=1 to n
        求出q(i) ┤        LET q(i)=q(i)+p(i,j)
                 │
                 └NEXT  j
                LET  y=y*q(i)
NEXT  i
```

以上各种程序段中出现的 a(i), b(i), r(i, j), s(i), p(i, j), q(i) 均为下标变量。

7.2 DO/LOOP 循环和 EXIT DO 语句

DO/LOOP 循环是一种比 FOR/NEXT 循环更为基本、更加灵活的循环结构。

7.2.1 带当型条件(WHILE <c>)的 DO/LOOP 循环结构

一般格式:

```
DO  WHILE  <c>
     ......┐
     ......┝ (循环体)
     ......┘
 LOOP
```

功能: 当条件<c>成立时, 重复循环体, 条件<c>不成立则结束循环。

其逻辑框图如图 7-3 所示。

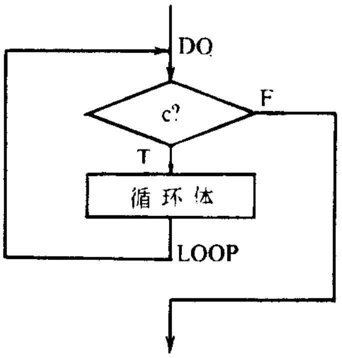


图 7-3 当型 DO 循环结构逻辑框图

执行 DO 语句即检测其右所带的 WHILE <c>, 当<c>成立时则执行循环体。LOOP 语句使程序返回到 DO 语句, 再检测条件<c>是否成立。如此反复, 直到<c>不成立时结束循环。因此, 条件<c>应在循环体中变化。

例 5 计算单位圆内一系列内接正 n 边形($n=3, 4, 5, \ldots, k$)的面积 s:

$$s = n\times\sin(\pi/n)\times\cos(\pi/n)$$

程序:

```
INPUT  k
LET  n=3
PRINT  "n", "s"
DO  while n<=k
    LET  s=n*sin(pi/n)*cos(pi/n)
    PRINT  n, s
    LET n=n+1
LOOP
END
```

7.2.2 带直到型条件(UNTIL <c>)的 DO/LOOP 循环结构

一般格式:

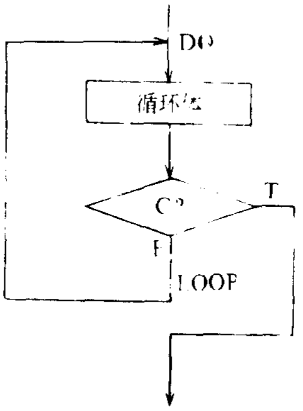


图 7-4 直到型 DO 循环结构逻辑框图

```
DO
    ......
    ......  } (循环体)
    ......
LOOP  UNTIL <c>
```

功能: 当条件<c>不成立时, 重复循环体, 条件<c>成立则结束循环。

其逻辑框图如图 7-4 所示。

进入此循环即执行循环体, 执行 LOOP 语句时即检测其右所带的 UNTIL <c>。当<c>不成立时, 返回到 DO 语句, 再执行循环体, 到 LOOP 处再检测条件<c>是否成立。如此循环, 直到<c>成立时结束循环。显然, <c>也应在重复循环体时发生变化。

例 6 用台劳级数求 e 的近似值, 直到最后一项小于 10^{-10}。

$$e=2+\frac{1}{2!}+\frac{1}{3!}+\frac{1}{4!}+\cdots\cdots$$

程序:

```
LET e, i, t=2
DO
    LET e=e+1/t
    LET i=i+1
    LET t=t*i
LOOP until (1/t)<10e-10
PRINT "e="; e
END
```

7.2.3 EXIT DO 语句

语句格式:

EXIT DO

功能: 执行此语句则退出最近一层 DO/LOOP 循环结构。

例 7 已知几何级数之首项为 2, 公比为 2, 计算并输出此级数及项数, 直到出现大于 1024 的项。

程序:

```
LET a, r=2
LET i=1
DO
```

```
      PRINT  i; a,
      LET  a=a*r
      IF  a>1024 then  EXIT  DO
      LET  i=i+1
   LOOP
   END
```

EXIT DO 语句使 DO/LOOP 循环增强了灵活性，从而方便算法的表达。由于它的退出仅仅涉及该层循环结构，因此并不影响其他程序模块。它既具有非结构化 BASIC 语言 GOTO 语句的灵活性，又防止了 GOTO 的随意性可能造成的程序逻辑混乱。

7.2.4 DO/LOOP 循环的其他四种形式

True BASIC 循环结构除了前述三种之外，还允许将当型条件 WHILE <c> 放在 LOOP 之右，也允许将直到型条件 UNTIL <c> 放在 DO 之右，从而可以构成另外几种不同的 DO/LOOP 循环。

(1) DO
……
…… } (循环体)
……
LOOP WHILE <c>

(2) DO UNTIL <c>
……
…… } (循环体)
……
LOOP

第(1)种循环一开始就执行循环体，到 LOOP 处检测当型条件，如果条件成立则重复。第(2)种开始即检测直到型条件，当条件不成立时重复。

(3) DO WHILE $<c_1>$
……
…… } (循环体)
……
LOOP UNTIL $<c_2>$

(4) DO UNTIL $<c_1>$
……
…… } (循环体)
……
LOOP WHILE $<c_2>$

第(3)种循环开始即检测当型条件 $<c_1>$，成立则执行循环体，到 LOOP 处又检测直到型条件 $<c_2>$，不成立则返回到循环起点再检测条件 $<c_1>$。

第(4)种循环开始检测直到型条件 $<c_1>$，不成立时执行循环体，到 LOOP 处检测当型条件 $<c_2>$；成立则返回到循环起点，再检测条件 $<c_1>$。

以上 $<c>$，$<c_1>$，$<c_2>$ 是逻辑表达式。每次执行循环，无论是 DO 右边的条件或 LOOP 右边的条件均被检查，然后决定是否重复执行循体。对于 WHILE 条件，当成立时则重复，否则退出循环。对于 UNTIL 条件，当成立时则退出循环，否则重复。关于循环，最关键问题之一是“如何建立循环”。一般，需要由所给问题的性质及对该问题进行处理所采用的算法来决定。另一个问题是“建立循环后，要能确保其正确地结束”。在程序中出现不能结束的无限循环，是经常出现的错误。其原因通常是没能正确地设立循环检测条件，或在循环过程中无法达到循环的终止条件。

使循环终止的最基本要求是：每重复执行一次循环体，就必须使与终止条件有关的一个或多个变量的值发生变化，在经过有限次重复之后，循环的检测条件就能够从成立变到不成

立（或者相反）。因此，设计循环时，应正确地选择循环变量，并使其在循环过程中应向循环终止的方向发生相应的变化。

7.3 程序举例

例 8 求阶乘和 $m=\sum_{n=1}^{20} n!$。

程序(1):

```
LET  m=0
LET  t=1
FOR  n=1 to 20
     LET  t=t*n
     LET  m=m+t
NEXT  n
PRINT  m
END
```

程序(2):

```
LET  m=0
LET  t、n=1
DO  WHILE  n<=20
     LET  t=t*n
     LET  m=m+t
     LET  n=n+1
LOOP  UNTIL  n>20
END
```

例 9 对裴波纳奇(Fibonacci)数列:

1，1，2，3，5，8，13，21，34，55，89，…

计算前 *n* 项的值。

从数列的第 3 项开始，每项都是其前 2 项之和，即 $F_i=F_{i-2}+F_{i-1}$。

程序(1):

```
INPUT  n
LET  f1=1
LET  f2=1
PRINT  f1、f2,
FOR  i=3 to n
     LET  f=f1+f2
     PRINT  f,
     !
     LET  f1=f2
     LET  f2=f
NEXT  i
END
------------------------
Ok.RUN
? 12
1         1         2         3         5
8         13        21        34        55
89        144
Ok.
```

程序(2):

```
INPUT  n
LET  f1，f2=1
PRINT  f1，f2,
LET  i=3
DO  WHILE  i<=n
     LET  f=f1+f2
     PRINT  f,
     LET  i=i+1
     LET  f1=f2
     LET  f2=f
LOOP
END
```

例 10　对多项式

$$Y = a_0X^n + a_1X^{n-1} + a_2X^{n-2} + a_3X^{n-3} + \cdots + a_{n-1}X^1 + a_n$$

求值。

按“秦九韶法”将上式改写为:

$$y = (\cdots(a_0x+a_1)x+a_2)x+\cdots+a_{n-1})x+a_n$$

其规律是反复进行赋值处理:

$y_1 \leftarrow a_0x+a_1$, $y_2 \leftarrow y_1x+a_2$, $y_3 \leftarrow y_2x+a_3$, $\cdots$, $y_{n-1} \leftarrow y_{n-2}x+a_{n-1}$, $y_n \leftarrow y_{n-1}x+a_n$

程序(1):

```
DATA (a0~an个系数数值)
INPUT  n, x
READ  y
FOR  i=1 to n
     READ  a
     LET  y=y*x+a
NEXT  i
PRINT  y
END
```

程序(2):

```
DATA (a0~an个系数数值)
INPUT  n, x
READ  y
LET  i=1
DO
     READ  a
     LET  y=y*x+a
     LET  i=i+1
LOOP  until  i>n
PRINT  y
END
```

例 11　统计全校 30 个班(每班人数不等, 每个学生考 3 门课), 每个班每门课的平均成绩和各门课的全校总平均成绩。

程序:

```
LET  x, y, z, n0=0
FOR  c=1 to 30
     READ  n
     LET  x1, y1, z1=0
     FOR  k=1 to n
          READ  s, l, h
          LET  x1=x1+s
          LET  y1=y1+l
          LET  z1=z1+h
     NEXT  k
     PRINT  x1/n, y1/n, z1/n
     LET  x=x+x1
     LET  y=y+y1
     LET  z=z+z1
     LET  n0=n0+n
NEXT  c
PRINT  x/n0, y/n0, z/n0
```

```
! 30 个班每班的人数及三门课的成绩数据
DATA (n1, s1, h1, l1, s2, h2, l2, …, 某班人数及成绩 3×n1+1 个数据)
DATA (n2, …)
  .
  .
  .
END
```

例 12 用迭代法求方程近似根。

设给定方程 $f(x)=0$，其中 $f(x)$为连续函数。

将方程改写为等价的，便于迭代的形式：

$$x=g(x)$$

当 $g(x)$满足条件：$|g'(x)|\leqslant q$（q 为小于 1 的正数），则存在收敛于方程根的近似根序列 x_0、x_1，x_2，…，x_n，…（x_0 为初始值）：

$$x_1=g(x_0),\quad x_2=g(x_1),\quad x_3=g(x_2),\quad \cdots,\quad x_n=g(x_{n-1}),\quad \cdots$$

这是一个无限过程，当迭代有限步后若相邻的两近似根之差的绝对值小于规定的误差则停止迭代。

若 $f(x)=x^3-x^2-1$，求在 $x=1.5$ 附近的根。

由 $f(x)\rightarrow g(x)$的过程，令 $f(x)=0$ 即 $x^3-x^2-1=0$，作恒等变换

$$x^3=x^2+1\rightarrow x=1+1/x^2$$

$$g(x)=1+1/(x\times x)$$

程序：

```
INPUT  prompt "x0, e=?": x0, e      ! 输入初始近似根x0及允许误差e
LET  x1=1+1/(x0 * x0)
DO  while  abs(x1-x0)>e
     LET  x0=x1
     LET  x1=1+1/(x0 * x0)
LOOP
PRINT  x1
END
-------------------------------------
Ok.RUN
x0, e=? 1.5, 0.00001
 1.46558
Ok.
```

例 13 求出最小的 100 个素数。

数 2 和 3 显然是素数。程序检验从 5 开始的奇数是否是素数，直到素数的个数等于 100 时结束。

程序：

```
PRINT  2, 3,
LET  n=2
LET  k=3
```

```
DO
    LET  k=k+2
    FOR  i=3  to  k-1                          ! 判断数 k 是否为素数
        IF  mod(k, i)=0  then  EXIT  FOR
    NEXT  i
    IF  i=k  then
        PRINT  k,                              ! 输出素数 k 并计数
        LET n=n+1
    END  IF
LOOP  until  n=100                             ! 当 n=100 时退出循环
END
```

* **例 14**　用梯形法求定积分

$$I=\int_a^b f(x)\mathrm{d}x$$

的近似值 I_n，并给出误差限 E_0。

梯形法求定积分的基本方法是将积分区间$[a,\ b]$分割成 n 个小区间，每个小区间的长度为 $h=(b-a)/n$，称为步长。以区间端点 x_{i-1}，x_i 的函数值 $f(x_{i-1})$和 $f(x_i)$连线的直线段来近似原函数的弧线段，以直边梯形面积来近似小区间上的积分(曲边梯形面积)，即：

$$\int_{x_{i-1}}^{x_i} f(x)\ \mathrm{d}x\approx\frac{h}{2}\ 〔f(x_{i-1})+f(x_i)〕$$

将 n 个小区间上的近似积分相加得梯形法求积分公式

$$T_n=\frac{h}{2}\ 〔f(a)+2\sum_{k=1}^{n-1}f(x_k)+f(b)〕$$

梯形法的截断误差为：

$$E=\frac{-h^2}{12}(b-a)f''(\xi),\ a<\xi<b$$

在实际使用时根据积分精度要求，可按截断误差公式确定步长 h。也可以依据不同的 n 值导出不同的 h 值，再由误差公式算出相应的误差限。

设计算积分：

$$I=\int_0^1\frac{4}{1+x^2}\mathrm{d}x,\qquad h=10^{-3}\qquad（\ \mathrm{I}\ 的精确值为\ \pi\ ）$$

被积函数 f(x)为$\frac{4}{1+x^2}$。计算截断误差需要用到 f(x) 的二阶导数：$f''(x)=-\frac{2-6x^2}{(1+x^2)^3}$。为了确定 $f''(\xi)$，在区间 [0，1]上寻找 $f''(x)$ 的最大值作为 $f''(x)$的上界。利用此上界计算出相应的误差限 E_0。

程序：

```
LET  a=0
LET  b=1
DO
   INPUT  prompt"n=?": n
```

```
IF  n=0 then EXIT DO
LET  h=(b-a)/n
LET  x=a
LET  fa=4/(1+a^2)
LET  fb=4/(1+b^2)
LET  f=0
FOR  k=1 to n-1
     LET  x=x+h
     LET  f0=4/(1+x^2)
     LET  f=f+f0
NEXT  k
LET  tn=h/2*(fa+2*f+fb)
PRINT  "In=";tn
LET  x=a
LET  r0=abs(-(2-6*x^2)/((1+x^2)^3))
FOR  x=a to b step 0.01
     LET r=abs(-(2-6*x^2)/(1+x2)3))
     IF  r>r0 then LET r0=r
NEXT  x
LET  e0=abs(-h*h/12*(b-a)*r0)
PRINT  "E0=";e0
LOOP
END
------------------------
Ok.RUN
n=? 100
In=3.15158
E0=1.66667e-5
n=? 0
Ok.
```

程序也可以是先由给定的误差 *E*，依据误差估算公式导出符合要求的 *n*，然后算出积分的近似值。

程序：

```
LET  a=0
LET  b=1
LET  n=1
INPUT  prompt"E=?": e
INPUT  prompt "a1, b1, c1=?": a1, b1, c1
LET  x0=a1
LET  r0=abs(-(2-6*x0^2)/((1+x0^2)^3))
FOR  x0=a1 to b1 step c1
     LET  r=abs(-(2-6*x0^2)/((1+x0^2)^3))
     IF  r>r0 then LET  r0=r
NEXT  x0
! r0为区间〔a1, b1〕上按步长c1找到的f″(x)之最大值
LET  x=a
DO
     LET  n=2*n
     LET  h=(b-a)/n
     LET  e0=abs(-h*h/12*(b-a)*r0)
LOOP  until  e0<e        ! 根据给定误差e求出对应的n
LET  fa=4/(1+a^2)
LET  fb=4/(1+b^2)
LET  f=0
FOR  k=1  to  n-1        ! 根据n计算符合误差e要求的In
     LET  x=x+h
```

```
          LET  f0=4/(1+x^2)
          LET  f=f+f0
NEXT  k
LET  in=h/2*(fa+2*f+fb)
PRINT  "In="; in, "(n="; n; ")"
END
```

```
-----------------------------------------
Ok.RUN
E0=? 0.00001
a1, b1, c1=? 0, 1, 0.1
In=3.1455          (n=256 )
Ok.
```

习 题 7

(1) 用 DO / LOOP 循环改写下列程序。

```
FOR  k=m to n
        LET  s=s+k
        IF  s>abs(m+n)  then
             EXIT  FOR
        END  IF
NEXT  k
END
```

(2) 试计算下列循环体的执行次数，并求 x 的值。

```
①LET  x=1          ②LET  x=0          ③LET  x=0           ④LET  x=-1
 DO WHILE x>0       DO UNTIL x<0       DO WHILE x<1        DO UNTIL x>0
      LET x=x-1          LET x=x-1          LET x=x+1           LET x=x+1
 LOOP UNTIL x<0     LOOP WHILE x>0     LOOP UNTIL x>0      LOOP WHILE x<0
```

(3) 编程计算下列各式，直到第 n 项的值小于 10^{-4} 时止:

① $\sum\limits_{K=1}^{n}\frac{1}{k(k+1)}$　② $\sum\limits_{k=2}^{n}\frac{K}{k^2-1}$　③ $\sum\limits_{k=1}^{n}\frac{\sin k}{k}$

(4) 百鸡问题：鸡翁一，值钱五；鸡母一，值钱三；鸡雏三，值钱一；百钱买百鸡，问鸡翁，鸡母，鸡雏各几何？本题有多解，试编一程序求出所有的答案。

(5) 将本金10万元，以年利7分5厘存款，试计算1，2，…，10年后，以单利存款时本利合计是多少，以复利存款时本利合计是多少？

(6) 计算1到100中，既不能被4也不能被5整除的所有整数之和。

(7) 写一程序，寻找被 2，3，5除时余数是1的最小的10个数。

(8) 试从 1～50的整数中，求奇数之和与偶数之和。

(9) 求整数1～999中的所有素数之和。

提示：整数N为素数的充分条件：当$1<K<\sqrt{N}$时，N不能被K整除。

(10) 已知$y=3x^5-6x^4+14x^2-7x+100$，当x在从0到2的范围内，以步长 0.1增加时，试求 y 的最大值和最小值。

(11) 数列A由N个数组成，已知它的前5项为 a，b，c，d，e，从第6项起，每项都是其前面 5 项之和，求数列 A。

(12) 已知：$\sin^{-1}x = x + \sum_{n=1}^{\infty}(\frac{x^{2n+1}}{2n+1}(\prod_{k=1}^{n}\frac{2k-1}{2k}))$，求$\sin^{-1}0.5$的近似值($\varepsilon < 10_{-4}$)。

(13) 计算积分 $\int_{b}^{a} e^{-x}\mathrm{d}x$。 (14) 读入正整数$M$，$N$，求$M$和$N$之间的所有完全数(所有因子之和等于其本身的数为完全数。例如：1，2，4，7，14 为 28 的因子，而 1+2+4+7+14 = 28，故 28 是完全数)。

(15) 输出由 1，2，3，4这4个数字组成的全排列（千，百，十，个位数上的数字不可重复）。

(16) 读入一行英文语句，试将其中的单词按输入顺序分行印出。

(17) 设英文字母按a→z的顺序以01→26编码。若字母L_1的编码为M_1，则将L_1变换为字母 L_2，使得 L_2 的编码 $M_2 = \mathrm{MOD}(M_1 \times i+j, 26)$（其中， i，j 为正整数）。编写程序，读入 i，j，然后对输入的每一行英文句子，将其中的字母按上述规则转换成密码输出，但保留单词原有的空格符和其它符号，直到输入空语句为止。

(18) 编写程序，读入整数i，j，然后对上题输出的密码进行解密。

(19) 人民币的面值有 100，50，10，5，2，1元； 5，2，1角以及 5，2，1分等12种。试编程序计算，对于任一数额的钱款，应如何以不同面值的钞票支付，使得所用钞票张数最少。

第8章 数组与矩阵

在前面的程序设计中频繁地用到简单变量、表达式和函数。它们的一个重要特点是其值不可再分(数学上称之为纯量或标量)。但是在用计算机处理实际问题时，常常要求考虑由一些个体数据组成的数据集合。例如：以班级为单位学生某门课的考试分数（一个学生的考分是简单个体，而全体学生的考分则为简单个体的集合）；矩阵（矩阵的元素是个体，矩阵是元素个体的集合)；数据表格（表格中的每项数据是个体，而整个表格则是数据的集合）等等。它们都由多个分量组成，数据之间具有一定的相互关系。

一般，把数据的组织形式(数据个体间的结构关系)称为数据结构。本章介绍的数组就是一种最常用的数据结构(一种均匀的线性结构，在计算机中以一组连续的内存单元来存贮这种类型的数据)。数组在程序设计中占有非常重要的地位。

True BASIC 不仅可以定义和使用多维数组，而且对于典型的数组运算，还设有专门的矩阵语句、矩阵的内部数组和矩阵函数。

8.1 数组说明与数组遍历

数组由同名下标变量构成，它的分量(或元素)的次序由下标来表示。它能有效地表示具有某种共同特性的相关数据。

数组作为一个同时存有许多数值(或字符串值)的变量，有时为了应用的需要，常将一维数组称为向量、二维数组称为矩阵、行列式等。

8.1.1 数组说明语句

语句格式:

DIM array(min:max,...)

功能：指定数组的名字、维数以及每一维的下界值和上界值。

其中，array 数组名应是一个合法的变量(数值变量或字符串变量)名，各维的下界值 min 和上界值 max 应是数值常量。如果不写出下界值，并连同冒号“:”一起省略，则默认为 1。程序所用的数组可以由多个 DIM 说明，也可以由一个 DIM 语句说明多个数组，这时被说明的数组之间要以逗号分隔。

数组的说明(又称为数组的定义)必须先于数组的使用。因此，宜在程序的开始部分就说明程序中所用的数组。

数组的维数可从 1 维至 n 维(IBM-PC True BASIC 的 $n<255$)，每维的下界和上界可以取$-1\times10^9\sim+1\times10^9$间的整数。说明下上界时不能用变量或表达式。数组某一维的下界和上界所限定的整数值范围，称为该维的界域，每一维的下标必须取界域内的整数。数组的维数及界域规定了数组的型态和大小。数组的维数及界域一经确定，下标变量的个数便随之确定。

数组经 DIM 语句说明之后，True BASIC 系统便在内存中开辟相应的连续存贮单元提供给数组中的各个下标变量使用。一般、一个数组对应一组连续的单元。数值数组下标变量的初值为 0，字符串数组下标变量的初值为空串。

8.1.2 一维数组

只带单下标的数组称为一维数组。例如，数组说明语句:

```
DIM  A(11)
```

它指定了一个名为 A 的一维数组，其下标的上界为 11。即该数组由 11 个下标变量 A(1)，A(2)， … ，A(10)，A(11)组成。未给出界域下界值时，下界为 1。上面的数组说明也可以写成: DIM A(1:11)，同时给出了下界值 1 和上界值 11，中间以冒号“:”分隔。

又例如:

```
DIM  name$(36), x(-5:5)
```

则分别说明了名字为 name$ 和 x 的两个数组，前者为字符串型数组，由 36 个字符串下标变量(name$(1)，name$(2)， …， name$(35)，name$(36))组成; 后者为数值型数组，共 11 个数值下标变量(x(-5)，x(-4)，… ，x(0)，x(1)， …，x(5))。

例 1 计算 10 个数据 A_1，A_2，…，A_{10} 的加权平均数 M:

$$M = (q_1 \times A_1 + q_2 \times A_2 + \ldots + q_{10} \times A_{10}) / 10$$

其中，q_i 为第 i 项数据 A_i 的权系数。

程序:

```
DIM  a(10), q(10)
FOR  i=1 to 10
     READ  a(i)
NEXT  i
FOR  j=1 to 10
     READ  q(j)
NEXT  j
LET  m=0
FOR  k=1 to 10
     LET  m=m+q(k)*a(k)
NEXT  k
PRINT  "M=";m/10
DATA  20, 40, 30, 50, 80, 90, 70, 60, 20, 10
DATA  0.7, 0.5, 0.6, 0.4, 0.1, 0.2, 0.3, 0.8, 0.1, 0.1
END
```

在这个例子中，程序首先用 DIM 语句说明了数组 A 和 Q。数组 A 的 10 个下标变量 A(1)、A(2)，…，A(10)用来存放数据 A_1，A_2，…，A_{10}; 数组 Q 的 10 个下标变量 Q(1)，Q(2)，…，Q(10)则用来存放数据 q_1，q_2，…，q_{10} 的数值．由于 A(i)及 Q(j)所代表的下标变量是由下标 i，j 的具体取值来决定的，因此读数据时，借助循环变量 i 的取值变化来对 A(1)，…，A(10)进行读数; 借助循环变量 j 的取值变化来对 Q(1)，…，Q(10)进行读数。按公式进行统计运算时，又利用循环变量 k，依次将 Q(1)*A(1)，Q(2)*A(2)，…，Q(10)*A(10)累加存入变量 M 之内。由于数组元素的下标可以是一个表达式，因而使这种有规律的运算能够简练地表达出来。

8.1.3 二维数组

带有双下标的数组称为二维数组。例如，数组说明语句

DIM　B(3，4)

说明了一个名字为 B 的二维数组，它的第一维下标的上界值为 3，第二维下标的上界值为 4．该数组由 3×4＝12 个双下标变量

```
B(1,1)    B(1,2)    B(1,3)    B(1,4)
B(2,1)    B(2,2)    B(2,3)    B(2,4)
B(3,1)    B(3,2)    B(3,3)    B(3,4)
```

组成，又称为有 3 行 4 列元素。

类似于一维数组说明，未给出维的下界值则表示下界值为 1．因此上面的数组说明与 DIM　B(1:3,1:4)是等价的。

又例如：

DIM　Seat(31,36)，P＄(0:10,0:20)，Item＄(3,3)

则分别说明了名为 Seat，由 31 行 36 列元素组成的二维数值数组；名为 P＄，由 11 行(包括第 0 行)21 列(包括第 0 列)元素组成的二维字符串型数组；名为 Item＄，有 3 行 3 列元素的字符串型数组。

类似地，也可以说明二维以上的数组．例如：

DIM　Book(10,25,30)，V(2,3,4,5)，R(2,2,2,2,2)

它们分别说明了 10 页 25 行 30 列的 Book 数组；2 册 3 页 4 行 5 列的 V 数组及 2 卷 2 册 2 页 2 行 2 列的 R 数组。

例 2　将 *m*×*n* 个数据置入一个 *m* 行 *n* 列的二维数组 *A* 之中．建立数组时，将第 1 个下标作为行，第 2 个下标当作列．假设 *m*＝3，*n*＝4，则有如下程序：

```
DIM a(3, 4 )                          ! 开辟二维数组
CLEAR                                 ! 清除屏幕
FOR  i=1 to 3
     FOR j=1 to 4
          READ  a(i,j)                ! 读第 i 行第 j 列元素的值
          PRINT  tab(i,4 * j);a(i,j)  ! 显示在屏幕的第 i 行第 4 * j 列上

     NEXT j
NEXT  i
DATA  1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12
END
```

在这个程序中，利用二重循环分别控制二维数组 A 的两个下标的变化．外层循环的循环变量 i 代人第一个下标的位置，内层循环的循环变量 j 代入第二个下标的位置．在循环过程中，对应于外层循环变量 i 的每一取值(1，2，3)，内层循环变量 j 均有 4 种取值(1，2，3，4)．因此总共可以产生出 3×4 种 i，j 的数值排列形式．这样便可以取尽数组 A 中所有可能的下标值，使得两层循环内部的 READ　A(i,j)语句能够对所有 3×4 个下标变量进行读数．程序运行后，二维数组 A 的各个下标变量的取值情况如表 8-1 所示。

如果交换内外层循环，程序变为以下形式：

```
DIM  a(3,4)
CLEAR
FOR  j=1 to 4
     FOR   i=1 to 3
```

```
          READ  a(i,j)
          PRINT  tab(i,4 * j); a(i,j)
      NEXT  i
NEXT  j
DATA  1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12
END
```

由于程序中两个下标变化的次序发生了变化，第 2 个下标 j 在外层，第 1 个下标 i 在内层，因此程序运行后二维数组 A 各下标变量的取值将变为表 8-2。

表 8-1　数组 A 取值(外层 i，内层 j)

J I	1	2	3	4
1	1	2	3	4
2	5	6	7	8
3	9	10	11	12

表 8-2　数组 A 取值(外层 j，内层 i)

J I	1	2	3	4
1	1	4	7	10
2	2	5	8	11
3	3	6	9	12

8.1.4　使用数组时应注意的一些问题

(1) 定义数组的维数与所用数组下标的个数必须一致。当出现不一致时，程序会出错，系统将给出如下出错信息：

wrong number of Dimensions (维数不对)

(2) 下标变量的下标值必须在该数组定义的界域之内，使用超出上、下界值的下标会导致程序出错。出错信息：

subscript out of bound (下标越界)

(3) 用 DIM 语句说明数组时，每个数组只能被说明一次。重复说明同一数组，将引起程序出错。出错信息：

name can′t be redefined (不能重新定义)

(4) 由于每个数组都按照各自的型态大小占用一定的计算机内存空间，因此说明数组时，应当合理地指定数组的个数、维数和界域。无论增加数组的维数或加宽数组的界域，都会引起数组元素个数的增加，所需的内存空间也随之急剧增大。例如，DIM A(5, 5, 5), B(0:5, 0:5, 0:5), C(5, 5, 5, 5)说明了 3 个数组，其中第 1 个数组具有 $5\times5\times5=125$ 个元素；第 2 个数组的界域比第 1 个数组加宽了五分之一，元素个数却增加了近五分之四，共有 $6\times6\times6=216$ 个；第 3 个数组比第 1 个数组只增加了一维，元素个数却是原来的 5 倍，变成 $5\times5\times5\times5=625$。又例如，一个三维数值数组，如果每维的界宽为 100 个元素，由于 True BASIC 中每个数值下标变量占用 8 个字节，那么该数组将占用约 8M 字节 ($1M=2^{20}=1048576$)。所以，在使用数组进行大数据量处理时：过多的维数或过宽的界域将使数组占据大量的内存空间。由于内存的有限性，可能会因内存空间不足而导致出错。出错信息为：

out of memory (超出内存)

由此可见，合理地开辟和使用数组，是提高程序空间效益的重要因素。

8.1.5　数组的遍历

上面两例，都是通过循环为整个数组读数，读数时分别对每一个下标变量进行读数操作。这种按某种顺序对所有下标变量逐个进行操作的过程，称为数组遍历。例 1 中使用一重循环来实现一维数组的遍历，例 2 中使用二重循环来实现二维数组的遍历。一般地，可以使用 n 重循环来实现 n 维数组的遍历，即每层循环分别控制一个下标的变化。

应当注意的是，遍历一个二维以上的数组时，对数组中每一下标变量依次逐个进行操作的顺序（遍历顺序），将由每层循环与所控下标的对应关系决定。例 2 中两个程序的对比说明，当内外层循环与所控下标的对应关系改变时，数组的遍历顺序也随之改变。

一般地，遍历数组时下标变化的规律是：越外层的循环所控制的下标变化得越慢，越内层的循环所控制的下标变化得越快。

例 3 在事务处理等应用问题中，常常需要根据各种数据表格进行数据的统计分析工作。下面的程序对给定的数据表格(表 8-3)按行、按列求和，并把和数置入该行、该列的末端。假设表格为 4 行 5 列，左上角的 3 行 4 列为待统计的原始数据。

表 8-3 原始数据表格

1	2	3	4	
5	6	7	8	
9	10	11	12	

程序按以下几个步骤设计：

(1) 首先将 3 行 4 列原始数据置入一个 4 行 5 列数组 B 的左上角。

(2) 然后对原始数据按行求和，将和数置入每行的第 5 列：

$$B(i,5)=\sum_{j=1}^{4}B(i,j),\qquad i=1,\ 2,\ 3$$

(3) 再对原始数据及新得出的第 5 列按列求和，将和数置入每列的第 4 行：

$$B(4,j)=\sum_{i=1}^{3}B(i,j),\qquad j=1,\ 2,\ 3,\ 4,\ 5$$

(4) 输出处理后的数组 B。

程序：

```
! 开辟数组并将数据读入数组的左上角
DIM  b(4,5)
FOR  i=1 to 3
     FOR j=1 to 4
         READ b(i, j)
     NEXT j
NEXT i
DATA  1,2,3,4,5,6,7,8,9,10,11,12
! 按行求和
FOR  i=1 to 3
     FOR j=1 to 4
         LET b(i,5)=b(i,5)+b(i,j)
     NEXT j
NEXT i
! 按列求和
FOR  j=1 to 5
     FOR i=1 to 3
         LET b(4,j)=b(4,j)+b(i,j)
     NEXT i
NEXT j
! 输出数组
FOR  i=1 to 4
     FOR j=1 to 5
         PRINT b(i,j);
     NEXT j
     PRINT
NEXT i
END
```

程序中的4个步骤，都是利用二重循环来完成对二维数组中下标变量的顺序操作。每个二重循环中每层循环与所控下标的对应关系有一定要求:

(1) DATA 语句的原始数据是按行排放的。读数时必须按行读，读完一行后再读下一行。因此读数时要求外层循环变量对应于行标，内层循环变量对应于列标；按行求和时，由于每行求和都是将前4列中的数据相加，同样须由外层循环控制行的变化，由内层循环控制列的变化；

(2) 按列求和时都是对每列的前3行中的数据相加，这时要求由外层循环控制列的变化，由内层循环控制行的变化；

(3) 由于 PRINT 语句(在不使用 TAB(x,y)函数的情况下)的输出只能由左向右、自上而下显示，为了正确地输出数据表格，应当从第一行起逐行显示每一列的数据。循环的外层和内层分别对应于数表的行与列。

8.2　数组的应用

数组元素的下标是有序的，因此对于次序有规律的数据的运算与处理问题，便可利用数组与循环来编制程序。数组方法是程序设计中解决有规律重复运算处理问题的有效手段。

下面的例题将说明数组的某些应用。

例4　将一组无序数据按从小到大的顺序进行选择排序(假设该组数据有20个数)。

排序步骤：首先将数据置入一维数组中；然后逐次将最小的数，第2小的数，第3小的数，…，直至最大数，分别调入数组的第1，第2，第3，…，直至最末一个元素中；最后将数组元素按下标顺序输出。为了实现将数据按大小顺序逐个调入对应的数组元素中，可逐步进行以下过程。

(1) 将第1个元素的值分别与第2到最后一个元素的值逐次进行比较。如果第1个元素的值大，则两元素之间进行数值交换；否则，继续下面的比较。逐次比较过程完毕后，第1个元素的值便是最小数。

(2) 将第2个元素的值分别与第3到最后一个元素的值逐次进行比较，如果第2个元素的值大，则两元素之间进行数值交换；否则，继续下面的比较。逐次比较过程完毕后，第2个元素的值便是第2小的数。

……

(i) 将第i个元素的值分别与第i+1到最后一个元素的值逐次进行比较，如果第i个元素的值大，则两元素之间数值作交换；否则，继续下面的比较。逐次比较完毕后，第i个元素的值便是第i小的数。

……

此过程只需执行到确定出倒数第2个元素为止，因为此时最后一个元素的值便是剩下的最大数。

程序:

```
DIM  x(20)                                  ! 开辟具有20个元素的数组
FOR  i=1 to 20                              ! 循环输入数据
     INPUT prompt "X"&str$(i)&"=?": x(i)    ! 输数时提示数据序号
```

```
NEXT  i
FOR  i=1 to 19                             ! 排出前19个数
     FOR j=i+1 to 20                       ! 每个数均与后面元素比较
         IF  x(i)>x(j) then                ! 如果大于后面元素则交换
             LET  t=x(i)                   ! 交换时需借助中间变量T
             LET  x(i)=x(j)
             LET  x(j)=t
         END  IF
     NEXT  j
NEXT  i
FOR  i=1 to 20                             ! 显示排序结果
     PRINT  x(i);
NEXT  i
END
```

如果将程序中的判断条件从“>”改为“<”，那么程序便可对数据按相反的顺序（即从大到小）进行排序。另外，如果将程序中的数值数组改为字符串数组，那么程序便可对字符串数据按 ASCII 码顺序进行排序。

例 5 根据一组实验观测数据，作线性拟合。

在科学研究中，经常需要对近似线性分布的观测数据点作出拟合直线方程，又称回归直线方程：

$$y(x)=ax+b$$

设观测数据为 (x_i, y_i)，$i=1, 2, 3, \ldots, n$。则每个数据点与拟合直线的偏差为

$$y(x_i)-y_i=ax_i+b-y_i, \quad i=1, 2, \ldots, n$$

偏差的平方和为

$$J(a, b)=\sum_{i=1}^{n}(ax_i+b-y_i)^2$$

根据最小二乘原理，解出 a，b：

$$a=\frac{\sum_{i=1}^{n}(x_i-\bar{x})(y_i-\bar{y})}{\sum_{i=1}^{n}(x_i-\bar{x})^2}$$

$$b=\bar{y}-a\bar{x}$$

其中

$$\bar{x}=\frac{1}{n}\sum_{i=1}^{n}x_i, \quad \bar{y}=\frac{1}{n}\sum_{i=1}^{n}y_i$$

设 $n=5$，观测数据为：

x_i	1	2	3	4	5
y_i	0	2	2	5	4

程序：

```
DIM  x(5), y(5)
```

```
LET  x0, y0=0
LET  n=5
FOR  i=1 to n
     INPUT  prompt ″X″&str$(i)&″, Y″&str$(i)&″=?″: x(i), y(i)
     LET  x0=x0+x(i)
     LET  y0=y0+y(i)
NEXT  i
LET  x1=x0/n
LET  y1=y0/n
LET  s, t=0
FOR  i=1 to n
     LET  s=s+(x(i)-x1)*(y(i)-y1)
     LET  t=t+(x(i)-x1)*(x(i)-x1)
NEXT  i
LET  a=s/t
LET  b=y1-a*x1
PRINT  ″a=″; a, ″b=″; b
END
```

结果得到: $a=11/10$;$b=-7/10$; $y(x)=1/10(10\times x-7)$。

在统计分析这类问题中，经常要对数值数据进行累加、累乘、求平均值等基本运算。熟练地掌握这些基本运算的程序对程序设计是十分方便的。

例6 求两个矩阵 A，B 的乘积 C. 由矩阵乘法规则得知，矩阵 A，B 能够进行乘法运算的条件是前一矩阵 A 的列数与后一矩阵 B 的行数相同. 乘积 C 为一矩阵，其行数等于 A 的行数，列数等于 B 的列数。若 A 为 n 行 p 列，B 为 p 行 m 列，则

$$\underset{n\times p}{\begin{bmatrix}A\end{bmatrix}}\ \underset{p\times m}{\begin{bmatrix}B\end{bmatrix}}=\underset{n\times m}{\begin{bmatrix}C\end{bmatrix}}$$

$$C_{ij}=\sum_{k=1}^{p}A_{ik}B_{kj}\qquad \begin{array}{l} i=1,\ 2,\ 3,\ \cdots,\ n \\ j=1,\ 2,\ 3,\ \cdots,\ m \end{array}$$

设 $n=3$， $p=4$， $m=2$

程序:

```
! 数组说明及参数设置
DIM  a(3,4), b(4,2), c(3,2)
LET  n=3
LET  p=4
LET  m=2
! 数组A读数
FOR  i=1 to n
     FOR  j=1 to p
          READ  a(i, j)
     NEXT  j
NEXT  i
DATA  1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12
! 数组 B 读数
FOR  i=1 to p
     FOR  j=1 to m
          READ  b(i,j)
     NEXT  j
NEXT  i
DATA  1, 2, 3, 4, 5, 6, 7, 8
! 求矩阵乘积 C
FOR  i=1 to n
     FOR  j=1 to m
          FOR k=1 to p
              LET  c(i,j)=c(i,j)+a(i,k) * b(k,j)
          NEXT  k
          PRINT  c(i,j);
     NEXT  j
     PRINT
NEXT  i
END
```

计算矩阵 C 时使用三重循环。其中 i、j 循环用于遍历 C 中的每一个元素，而每一个元素的值则通过 k 循环进行相乘相加计算．即：k 循环求出矩阵 C 的一个元素，j 循环求出矩阵 C 的一行元素，而 i 循环则求出矩阵 C 的所有各行，即全体元素．

例 7　显示杨辉三角形各行的数值．

可以利用二维数组的元素来表示杨辉三角形中的每一数字．数组的第 1 列 $A(i,1)$表示左边的腰,数组的主对角线元素$A(i,i)$表示右边的腰:

$$a(i,j)=1 \qquad , \quad j=1 \text{ 或 } j=i$$
$$a(i,j)=a(i-1,j-1)+a(i-1,j) \qquad , \quad j>1 \text{ 且 } j<i$$
$$(i=1,\ 2,\ 3,\ \ldots,\ n\ ;\ j=1,\ 2,\ 3,\ \ldots,\ i)$$

程序:

```
DIM  a(20,20)
CLEAR
FOR  i=1 to 20
     FOR  j=1 to i
          IF j=1 or j=i  then
             LET  a(i,j)=1
          ELSE IF  j>1 and j<i then
             LET  a(i,j)=a(i-1,j-1)+a(i-1,j)
          END  IF
          PRINT  tab(i,38-2 * i+4 * j); a(i,j)
     NEXT  j
NEXT  i
END
```

程序中利用二维数组 A 的下三角部分存放杨辉三角形。循环变量 i 表示行，j 表示列。

通过例子可以看出，数组的使用往往与循环结合在一起。数组具有可按编号灵活地调用和操作数据的性能，循环可实现规律性的重复运算，两者恰当配合是解决复杂程序设计问题的重要手段。

8.3 矩阵语句

矩阵是按行列顺序规则排列的数据集合。无论是工程计算、数据统计分析，还是事务处理，都经常使用矩阵形式的数据。在程序设计中，矩阵可直接以二维数组表示。

对于数组的整体运算，True BASIC 设置了矩阵(Matrix)语句(MAT 语句)，其主要的功能包括:

MAT READ	数组读数;
MAT INPUT	数组输入;
MAT PRINT	数组显示;
MAT …= …	数组赋值与矩阵代数运算;
MAT REDIM	数组大小调整。

MAT 语句虽然称为矩阵语句，但是它并不仅局限于对矩阵(二维数组)进行处理，而且能够有效地用于一维和多维数组。

8.3.1 数组整体读数

语句格式:

MAT READ array

作用: 将 DATA 语句中的数据一次性读入指定数组(array)。

例 8 读入一组姓名，再读入对应的年龄，输出姓名及相应的年龄。

程序:

```
DIM  name$(3),age(3)
MAT  READ  name$
MAT  READ  age
FOR  i=1 to 3
     PRINT  name$(i),age(i)
NEXT  i
DATA  Wang,Li,Zhang
DATA  26, 18, 22
END
```

```
-----------------------
Ok.RUN
Wang         26
Li           18
Zhang        22
Ok.
```

在这个程序中由 DIM 读语句说明了两个数组: 一个字符串数组 NAME$ 和一个是数值数组 AGE。程序运行时，通过两个 MAT READ 语句分别为这两个数组读数。因为读数时首先要一次性读完 NAME$ 的数据，然后再读 AGE 的数据，所以对应于同一数组的数据必须连续地放在一起，并按读数的顺序置入 DATA 语句中。

需要注意的是，当使用 MAT READ 语句为一个二维以上数组进行读数时，DATA 语句中的数据是按照数组左边下标变化慢，右边下标变化快的规律依次将数据读入各下标变量中的，这种变化规律称为“计数器”变化规律。

例如，对一个三维数组 A(1:10，1:10，1:10)，语句 MAT READ A 的作用等同于下面的循环过程：

```
FOR  i=1 to 10
     FOR j=1 to 10
         FOR k=1 to 10
             READ  A(i,j,k)
         NEXT  k
     NEXT  j
NEXT  i
```

此程序段中，下标 k 变化最快，j 次之，i 最慢。对于二维数组，读数过程是按行(而不是按列)进行的，即列变化快，行变化慢。

另外，在一个 MAT READ 语句中允许同时写出多个数组名。此时，将按照数组名出现的顺序先读入左边数组的数据后，再逐次向右读下一个数组的数据。因此例 1 中的

```
MAT READ NAME$
MAT READ AGE
```

可以写成下面的等效形式

```
MAT  READ  NAME$, AGE
```

8.3.2 数组整体输入数据

语句格式：

(1) MAT INPUT array

(2) MAT INPUT PROMPT expr$: array

作用：通过键盘将数据一次性输入到指定数组中。

输入数据时，各项数据之间以逗号“,”相隔。整个数组的数据输入完毕后按回车键表示结束。PROMPT expr$：用于显示对输入的提示，提示内容为 expr$。如果没有 PROMPT expr$：，则自动显示问号?，作为输入提示。

例 9 输入一组货物的单价及数量，输出对应的货物价值(单价×数量)。

程序：

```
DIM  p(5), q(5)
MAT  INPUT prompt  "Unit  price=?": p
MAT  INPUT prompt  "Quantity=?": q
PRINT  "Total price=";
FOR  i=1 to 5
     PRINT p(i) * q(i);
NEXT  i
END
```

```
----------------------------
Ok.RUN
Unit  price=? 1.25,3.74,0.8,12.03,5.55
Quantity=? 6,12,70,25,40
Total price= 7.5  44.88  56  300.75  222
Ok.
```

为数组整体输数时，如果输入的数据不够，计算机将继续询问数据。因此为数组输数也可以分几行输入数据。作分行输入时，每行数据后面应带一个逗号，表示数据尚未输入完毕，但最后一个数据之后不要加逗号。对于二维以上的数组，分行输入的方式是比较方便的。

例 10　为一个二维数组整体输数，按分行方式输入。

程序:

```
DIM  a(3, 4)
MAT  INPRT a
FOR  i=1 to 3
     FOR  j=1 to 4
           PRINT  using"###": a(i, j);
     NEXT  j
     PRINT
NEXT  i
END
```

```
-------------------------
Ok.RUN
? 1,2,3,4,5,6,7,8,
? 9, 10, 11, 12
1    2    3    4
5    6    7    8
9   10   11   12
Ok.
```

由此例可见，使用 MAT　INPUT 语句为二维以上的数组进行一次性输入时，数据也是按照“计数器”变化规律输入给各下标变量的。

此外，与 LINE　INPUT 语句对应的有 MAT　LINE　INPUT 语句。这一语句能够通过键盘，将多行字符串一次性输入指定的字符串数组中。它与 MAT　INPUT 语句的差别，仅仅在于它专用于输入字符串。在键盘输入过程中，每输入一个字符串后都必须按回车键表示该字符串结束，而逗号和双引号可以作为字符串本身的内容。因此，利用 MAT　LINE　INPUT 语句可以将一组常规格式的文章句子存入字符串数组中。

8.3.3　数组整体输出数据

语句格式:

```
MAT  PRINT  array
```

作用：将指定数组的所有元素的值，按顺序全部显示出来。

数组名后可以带逗号(,)或分号(;)。使用逗号(,)时，按标准格式显示；使用分号(;)时，按紧凑格式显示。数组名之后未带逗号或分号时，则默认显示格式为标准格式。

例 11　分别对数组 *A*，*B* 作整体读入与输出。

程序:

```
DIM  a(3,4), b(4,3)
MAT  READ  a, b
MAT  PRINT  a;
MAT  PRINT  b
DATA  1, 3, 5, 7, 9, 11, 13, 15, 17, 19, 21, 23
DATA  2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24
END
------------------------------------
Ok.RUN
  1    3    5    7
  9    11   13   15
  17   19   21   23
  2         4              6
  8         10             12
  14        16             18
  20        22             24
Ok.
```

8.3.4 数组整体赋值

语句格式:

MAT array=……

作用: 为指定的数组赋值。

若赋值号“=”的右边为一表达式，则求出表达式的当前值并赋给数组的每一元素；若它的右边为一数组，则将这个数组各元素的值按下标顺序的对应关系赋给指定数组（此时要求这两个数组的维数必须相同)。

例 12 分别对数组 *A*，*B*，*C* 作整体赋值与输出。

程序:

```
DIM  a(2,3), b$(2,3), c(2,3)
MAT  a=1
MAT  b$="B"
MAT  c=a
MAT  PRINT  a; b$, c;
END
```

```
------------------------
Ok. RUN
1   1   1
1   1   1
B             B             B
B             B             B
1   1   1
1   1   1
Ok.
```

MAT 赋值语句除可完成数组的整体赋值外，还以隐含的方式起调整数组的作用。将一完整的数组赋给指定数组时，如果指定数组中某一维的界宽不合适，那么这一维的下界将持不变，而上界会自行调整以适合赋值号右边数组的大小。因此赋值后两个数组对应元素值是相同的。但是，由于两个数组维数下界起始值可能不同，因而两个数组的对应元素各标值就可能不一致。

MAT 赋值语句赋值号的右边允许出现含单个运算符的矩阵代数表达式。此时，MAT 赋值语句将按照矩阵代数的运算规则求出表达式的矩阵结果，然后赋给指定数组。赋值时，同样会自动根据表达式矩阵结果的大小隐含地调整指定数组的界域尺寸。

假设 A，B，C 为数值矩阵，则 MAT 矩阵赋值语句能处理的矩阵运算有:

```
MAT  A=B+C
MAT  A=B-C
MAT  A=B*C
MAT  A=(expr)*B
```

其中，除矩阵乘法仅对一维和二维数组有效外，矩阵的加法、减法和数值乘矩阵(数值必须加圆括号)还适用于任意维数的数组。进行矩阵运算的两个数组的形态大小是相关的。对于加减法，要求相加减的两个数组必须形态大小相同，每维中只需有相同的界宽，而上下界可不一致。对于矩阵乘法，则要求前一数组的列数(即第二维的界宽)与后一数组的行数(即第一维的界宽)必须相同。

例 13 数组作加、减、乘运算时，界域间的关系。

程序:

```
DIM  a(2,3), b(2,3), c(3,2)
DIM  sum(2,3), dif(2,3), tim(2,3), mul(2,2)
MAT  READ  a, b, c
MAT  sum=a+b
MAT  dif=a-b
MAT  tim=(-2)*a
MAT  mul=a*c
MAT  PRINT  sum; dif; tim; mul;
DATA  1, 2, 3, 4, 5, 6
DATA  6, 5, 4, 3, 2, 1
DATA  1, 2, 3, 4, 5, 6
END
```

8.3.5 数组大小的调整

语句格式:

```
MAT  REDIM  array(min:max, …)
```

功能: 根据实际的数据量调整数组 array 的大小。

圆括号中的内容，重新说明数组调整后各维的新下界 min 和新上界 max 它们可以是任何数值表达式。如果调整数组时不需改变下界，则“min: ”部分可以省去。

例 14 任意给定平面上一组点的坐标，求出这组点中与原点的最远距离。

程序:

```
DIM  x(1), y(1)                              ! 开辟两个一维坐标数组
INPUT  prompt "how many points?": n          ! 提示输入实际点数
MAT  REDIM  x(n), y(n)                       ! 根据实际点数重调数组
FOR  i=1 to n                                ! 循环输入各点坐标
     PRINT  "X("; n; "), Y("; n; ")=";       ! 用 PRINT 产生输入提示
     INPUT  x(i), y(i)
NEXT i
FOR  i=1 to n                                ! 循环求最远距离 Dmax
     LET  d=sqr(x(i) 2+y(i)^2)
```

```
        IF  i=1 then
            LET  dmax=d
        ELSE
            LET  dmax=max(dmax,d)
        END  IF
    NEXT  i
    PRINT  "Max distance is"; dmax
    END
```

```
-------------------------------
Ok.RUN
How many points? 3
X( 1 ),  Y( 1 )=? 2,  6
X( 2 ),  Y( 2 )=? -7,  4
X( 3 ),  Y( 3 )=? 5,  3
Max  distance  is  8.06226
Ok.
```

需要注意：利用 MAT REDIM 语句重新调整数组后，并未清除数组中原来的内容，而是按照行的顺序将原来的内容在调整后的数组中重新排列：重排时，多出的数据被放弃，数据不足则补 0(数值数组)或补空串(字符串数组)。

除了 MAT REDIM 语句可以显式地调整数组大小外，MAT READ 和 MAT INPUT 语句还具有隐式调整数组的功能：利用 MAT READ 语句为数组读数或利用 MAT INPUT 语句为数组输数的同时，通过重新标出数组各下标的上下界来调整数组大小。

语句格式：

```
MAT  READ  array(min:max, … )
MAT  INPUT  array(min:max , … )
```

其中，下界 min 和上界 max 可以是任何数据表达式，标明方法与 MAT REDIM 语句相同。

例 15 利用 MAT READ 语句调整数组 *A*。

程序：

```
DIM  a(1, 1)                ! 定义 a 为 2 行 2 列的二维数组
READ  r, c                  ! 读入数组实际的行列数
MAT  READ a(r, c)           ! 调整数组 a 为 r 行 c 列并读数
FOR  i=1 to r
     FOR  j=1 to c
          PRINT  a(i, j);
     NEXT  j
     PRINT
NEXT  i
DATA  3, 4                  ! 数组为 3 行 4 列
DATA  1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12
END
```

上例中，首先使用 DIM 语句说明数组 A，确定数组 A 是二维的。然后通过一个 READ

语句读入标明数组实际大小的数据 R(行数)和 C(列数)，并将它们代入 MAT READ 语句中二维数组 A 后的括号内，从而实现在读数的同时调整数组大小为 R 行 C 列。

例 16 利用 MAT INPUT 语句调整数组 table。

程序：

```
DIM  table(1,1)                                  ! 开辟存放表格的二维数组
INPUT  prompt "R, C=?": r, c                     ! 输入数据表的实际行列数
MAT  INPUT  table(r,c)                           ! 调整数据表格的行列数
!                                                ! 并从键盘输入数据
FOR  j=1 to c                                    ! 在每列的位置上标出列号
     PRINT  tab(8*j); j;
NEXT  j
PRINT
FOR  i=1 to r                                    ! 逐行显示数据表格的内容:
     PRINT  i;                                   ! 标出行号
     FOR  j=1 to c                               ! 显示该行各列的数据
          PRINT  tab(8*j); table(i, j );
     NEXT  j
     PRINT
NEXT i
END
----------------------------------------
Ok. RUN
R, C=? 3, 4
?12, 56, 33, 62, 74, 18, 25, 50, 96, 4, 12, 70
        1       2       3       4
1       12      56      33      62
2       74      18      25      50
3       96      4       12      70
Ok.
```

另外，对于一维数组，使用 MAT INPUT 语句输入数据时可以将“?”代入数组后的括号中。这时数组的大小可以根据键盘上实际输入的数据个数作调整（下界固定，上界根据输入的数据个数变化），因此更加灵活。

例 17 利用 MAT INPUT 数组(?)语句，调整数组的大小，接受键盘输入的数组并输出和数。

程序：

```
DIM  a(1)                          END
MAT  INPUT  a(?)                   ---------------------------
FOR  i=1 to ubound(a)              Ok.RUN
     LET  m=m+a(i)                 ? 1,2,3,4,5
NEXT  i                            TOTAL = 15
PRINT  "TOTAL=";  m                Ok.
```

这个程序可求出任意多个数值的累加和，其中数组函数 UBOUND(A)是求一维数组 A 的上界值。

8.4 内部数组和矩阵函数

在 True BASIC 系统中还设置了一些性质固定的数组(表 8-4)，这些内部数组的内容是在使用一个数组之前，对数组进行初始化时常用的初始值。比如：全部为 0 的数组，全部为 1 的数组，主对角线为 1 其余为 0 的数组等。将内部数组用在 MAT 赋值语句赋值号的右边，可以对指定数组赋予某种形式的初值，同时还可根据需要重新对数组的大小进行调整。内部数组主要用于数组的初始化过程。

使用内部数组时，既可以带有明确的上下界，也可以不带上下界。当在 MAT 赋值语句赋值号的右边使用带有明确上下界的内部数组时，被赋值数组的大小将随之被调整，改变到与内部数组的上下界相一致。如果使用的是不带上下界的内部数组，那么赋值之后被赋值的数组将保持原来的大小。例如：

```
DIM  A(3,3)
MAT  A=ZER(4,4)
```

经过 MAT A=ZER(4,4)语句，数组 A 将被调整成 4×4 的数组（上界为 4，省略下界时默认下界为 1)，并且数组的每一元素值都置为 0(zero)，而

```
DIM  A(3,3)
MAT  A=ZER
```

是将 0 赋予数组 *A* 的每一元素，数组 *A* 保持原定义的大小，仍为 3 行×3 列。上述的 ZER(4,4)和 ZER 即为带上下界和不带上下界的内部数组。

注意：内部数组不能用于矩阵运算，但是可以与一个常数相乘。

表 8-4 内部数组

数组名	含　义
con	所有元素皆为 1 的数值数组。
con(m,n,…)	所有元素皆为 1，尺寸为 m×n... 的数值数组(下界的默认值为 1。亦可使用 to 或标出下界。下同)。
idn	单位矩阵(即主对角线元素为 1，其余元素为 0 的行数列数相等的二维数值数组)。
idn(n)	单位矩阵，尺寸为 n×n。
nul $	所有元素皆为空字符串的字符串数组。
nul $ (m,n，…)	所有元素皆为空字符串，尺寸为 m×n...的字符串数组。
zer	所有元素皆为 0 的数值数组。
zer(m,n，…)	所有元素皆为 0，尺寸为 x×y ×...的数值数组。

例 18 将一组学生年龄情况记录，按学生姓名的拼音字母顺序排列。

程序：

```
DIM   name $ (1),age(1)                     ! 开辟姓名和年龄数组
INPUT  prompt "How many students ?": n   ! 输入学生人数
MAT   name $ = nul $ (n)                    ! 利用空串数组调整姓名数组
MAT   age = zer(n)                          ! 利用全零数组调整年龄数组
```

```
! 读取姓名和年龄数据
FOR  i=1 to n
     READ  name$(i), age(i)
NEXT  i
! 排序
FOR  i=1 to n-1
     FOR  j=i+1 to n
          ! 对姓名的大写形式进行比较
          IF  ucase$(name$(i))>ucase$(name$(j)) then
             ! 交换姓名内容
             LET  n$ =name$(i)
             LET  name$(i)=name$(j)
             LET  name$(j)=n$
             ! 交换年龄内容
             LET  a=age(i)
             LET  age(i)=age(j)
             LET  age(j)=a
          END IF
     NEXT j
NEXT  i
! 显示排序结果
FOR  i=1 to n
     PRINT  name$(i), age(i)
NEXT  i
! 学生情况记录
DATA  Li ming, 23
DATA  Wang Gang, 25
DATA  Liu Ping, 20
END
```

程序运行时，首先输入学生人数 n，因为 DATA 语句中姓名和年龄数据是交错放置的，所以不能采取 MAT READ 语句的方法，而应该先利用内部数组调整 name$ 和 age 的大小，然后再利用循环混合读数。进行顺序比较时，利用 UCASE$ 函数将姓名的拼音字母统一转换成大写的形式，以适应姓名写成大写或小写的情况. 当调换姓名 name$(i)时，年龄 age(i)也应随之调换，这样便保持了同一记录数据的对应关系。

为了便于矩阵代数的运算，True BASIC 还设有 4 个矩阵函数，用来直接进行矩阵求逆、矩阵转置和求矩阵所对应的行列式的值(表 8-5)。

矩阵求逆对舍人误差非常敏感，所以使用 inv 函数之后应当检验 det 的值。如果其值近似于 0，则表明求逆的结果误差很大。

inv 和 trn 函数以矩阵作为函数值，因此应置于 mat 语句中合适的数组位置上；det、dot 函数的函数值是数值，因此可根据需要用在任意的数值表达式中。

例 19 求解线性方程组 $AX=B$，其中 A 为线性方程组的系数矩阵，B 为常数列向量。

若 C 为 A 的逆矩阵，则线性方程组的解向量 X 可表示为

$$X=CB \qquad 或 \qquad X=A^{-1}B$$

表 8-5　矩阵函数

函数名	作　　用
inv(A)	得到正方形矩阵 A 的逆矩阵。行列式值为 0 的方阵为奇异矩阵，如 A 奇异或近似奇异，则给出出错信息。
trn(A)	得到矩阵 A 的转置矩阵。若 A 为 M 行 N 列，则转置矩阵 R 为 N 行 M 列，其中，$R(i,j)=A(j,i)$。
det	得到最近一次用 inv 函数求逆的矩阵的行列式值。如果矩阵奇异或近似于奇异，则 det 的值为 0 或近似于 0。
det(A)	得到正方形矩阵 A 的行列式的值。
dot(A,B)	求出两个用一维数组表示的向量 A 和 B 的点积。这两个向量的分量个数必须相同，点积结果为一纯量数值。

下面的程序中，首先判断 A 的行列式的值是否为 0，如果为 0 则表明矩阵奇异，方程组无解；否则，求 A 的逆矩阵后再判断其行列式的值是否近似于 0，如果近似于 0 则提示求解的精度不可靠。

这是一个能够求解任意阶线性方程组的通用程序，方程组的系数及右端常数数据按顺序写入 DATA 语句中，程序运行时通过键盘回答方程组的阶数 n。程序中给出了一个三阶线性方程组的数据。

程序:

```
DIM  a(1,1), c(1,1), b(1), x(1)
INPUT  prompt "N=?":  n
MAT  READ  a(n,  n), b(n)
IF  det(a)=0  then
    PRINT  "No solution"
ELSE
    MAT  c=inv(a)
    IF  abs(det(a))<1e-8 then
        PRINT  "The solution is not reliable"
    END  IF
    MAT  x=c*b
    MAT  PRINT x;
END  IF
! 方程组的系数
DATA  3, 2, 1
DATA  1, 1, 1
DATA  2, 3, -1
! 方程组右端的常数
DATA  14, 10, 1
END
```

```
--------------------
Ok. RUN
N=? 3
 1   2   7
Ok.
```

此外，True BASIC 还有其他一些作用于数组的函数(表 8-6)。这些函数均以数值为函数值，可用在任意的数值表达式中。

例 20 根据需要临时定义一张二维数表，并通过键盘以填表的方式输入数据。运行时，首先输入表格各列的名称，接着输入各行的名称，确定表格的形式。然后，按表格对应位置输入数据。

表 8-6 数组函数

函数名	作用
size(A)	得出数组 *A* 中所有元素的个数。
size(A,n)	得出数组 *A* 第 *n* 维下标所有元素的个数，即第 *n* 维的界宽。
lbound(A)	得出一维数组 *A* 的下标下界。
lbound(A,n)	得出数组 *A* 第 *n* 维下标的下界。
ubound(A)	得出一维数组 *A* 的下标上界。
ubound(A,n)	得出数组 *A* 第 *n* 维下标的上界。

程序:

```
DIM  a(1,1), r$(1), c$(1)
CLEAR
MAT  INPUT  prompt ″ITEMS OF COL: ″: c$(?)          ! 输入各列的项目名称
MAT  INPUT prompt ″ITEMS OF ROW: ″: r$(?)           ! 输入各行的项目名称
MAT  a=zer(size(r$), size(c$))                      ! 按行列项目个数调整数组
FOR  i=0 to size(r$)                                ! 逐行输入表格各项
     FOR  j=0 to size(c$)                           ! 每行中逐列进行输入
          PRINT  tab(2*i+4, j*10+1);                ! 定位
          IF  i=0 and j>0 then                      ! 第 0 行显示列项目名
                PRINT  c$(j)
          ELSE  IF i>0 and j=0 then                 ! 第 0 列显示行项目名
                PRINT  r$(i)
          ELSE  IF i>0 and j>0 then                 ! 其他位置上输入数据
                INPUT  prompt ″ ″: a(i,j)
          END  IF
     NEXT  j
NEXT  i
! 此处可加入处理表格的程序段
END
------------------------------------------------------------
Ok. RUN
ITEMS OF COL: SPRING, SUMMER, AUTUMN, WINTER
ITEMS OF ROW: 1985, 1986, 1987, 1988
              SPRING     SUMMER     AUTUMN     WINTER
1985          12.5       15.8        ...        ...
1986          ......
1987          ......
1988          ......
```

程序采用 size 函数分别求出 r $ 和 c $ 的项数，并以此作为行列数来确定表格的大小。

8.5 程序举例

例 21 拉格朗日插值。

假设在实验中取得 n 组数据(x_i, y_i)，利用拉格朗日插值公式可构造出这组数据的插值函数(函数曲线通过已知数据点)。利用这个函数可以计算已知点以外的点的函数值，研究被插值函数的性质等。

根据拉格朗日插值公式:

$$L(x)=\sum_{i=1}^{n} y_i \prod_{j=1,\ j\neq i}^{n} \frac{(x-x_j)}{(x_i-x_j)}$$

对于任意输入的插值点 x，计算出相应的 $L(x)$值。

程序:

```
DIM  x(1),y(1)
READ  n
MAT  x=zer(n)
MAT  y=zer(n)
FOR  i=1 to n
       READ x(i),y(i)
NEXT  i
INPUT  prompt "x=?": u
LET  v=0
FOR i=1 to n
      LET  t=1
      FOR  j=1 to n
            IF  i<>j  then
                 LET  t=t*(u-x(j))/(x(i)-x(j))
            END  IF
      NEXT  j
      LET  v=v+y(i)*t
NEXT i
PRINT  "L("; u; ")="; v
DATA  5                                               !点数
DATA  1, 3.5, 2, 4.8, 3, 4.4, 4, 5.1, 5, 3.9          ! xi, yi点对的数据
END
```

```
----------------------------------
Ok.RUN
x=?2.5
L(2.5)=4.50156
Ok.
```

例 22　冒泡排序法。

先将待排序的 n 个数据置入一维数组 x 中，然后，对第 2 至第 n 个下标变量按顺序分别作如下处理:

把它与上一个(即下标值小 1 的一个)下标变量作比较，如果它比上一个变量小则与其互换。交换后继续与上面的一个变量作比较。这种比较与交换的过程一直向上进行，直至出现大于等于的情况或达到顶部为止。

冒泡排序法的排序过程，即是把第 2 至第 n 个数逐个地冒升到所有比它大的数之上，从而实现排序的。

程序:

```
DIM  x(1)
INPUT  prompt  "n=?": n
MAT  READ  x(n)
FOR  i=2 to n                                          ! 逐个向上冒升
     FOR  i=i-1 to 1 step  -1                          ! 冒升直到顶部
          IF  x(j+1)>=x(j)  then  EXIT FOR             ! 大于等于则停止
          LET  t=x(j)                                  ! 否则升到上一位
          LET  x(j)=x(j+1)                             ! (与上一位交换)
          LET  x(j+1)=t
     NEXT  j
NEXT  i
MAT  PRINT  x;
DATA  150,3.5,-6,89,45,2,45,0,50,-100,100
END
----------------------------------
Ok.RUN
n=?10
-100  -6  0  2.45  3.5  45  50  89  100  150
Ok.
```

由于每次使数据冒升的比较和交换过程并不一定到达顶部，因此冒泡排序法与本章例 4 选择排序法相比，数据的比较和交换次数较少，排序速度较快。

例 23　表格数据的统计。

本例程序处理一张学生成绩表。程序运行时，从键盘输入学生人数和平时、段考、期考 3 项成绩在总成绩中所占的百分比。程序将自动算出每个学生的总成绩，以及全班各项成绩的平均值。

程序:

```
DIM  n$(1), s(1, 4), p(3)
```

```
INPUT  prompt ″How many students?″： n
MAT  n$ = nul$(n)
MAT  s=zer(n+1，4)
! 输入平时、段考、期考成绩占总成绩的百分比
DO
     INPUT  prompt  ″Exercise(%)： ″： p(1)
     INPUT  prompt  ″1st test(%)： ″： p(2)
     INPUT  prompt  ″2nd test(%)： ″： p(3)
LOOP  until  p(1)+p(2)+p(3)=100
! 显示表头
PRINT
PRINT  ″Name″，″Exercise 1st test 2nd test Final″
PRINT  repeat$(″-″，55)
! 按行显示各学生的成绩
FOR i=1 to n
     READ  n$(i)
     PRINT  n$(i)，
     ! 显示平时、段考、期考成绩
     FOR j=1 to 3
          READ s(i，j)
          ! 数据按保留一位小数的格式输出
          PRINT  using ″######.# ″： s(i，j)；
          LET  s(i，4)=s(i，4)+p(j) * s(i，j)/100
          LET  s(n+1，j)=s(n+1，j)+s(i，j)
     NEXT  j
     ! 显示总评成绩
     PRINT  using ″   ####.#″： s(i，4)
     LET  s(n+1，4)=s(n+1，4)+s(i，4)
NEXT  i
! 最后按行显示各项成绩的平均值
PRINT  repeat$(″-″，55)
PRINT  ″Average=″，
FOR j=1 to 4
     PRINT  using ″######.# ″： s(n+1，j)/n；
NEXT  j
! 每个学生的姓名和成绩数据
DATA  Li Ming，78，62.5，94
DATA  Wang Ping，85，98，96.5
DATA  Zhang Qiang，70，85，67
END
-------------------------------------------
Ok.RUN
```

```
How many students? 3
Exercise(%):  15
1st test(%):  30
2nd test(%):  55
Name                 Exercise   1st test   2rd test   Final
-------------------------------------------------------------
Li Ming              78.0       62.5       94.0       82.1
Wang Ping            85.0       98.0       96.5       95.2
Zhang Qiang          70.0       85.0       67.0       72.8
-------------------------------------------------------------
Average=            77.7       81.8       85.8       83.4
Ok.
```

例 24 计算平面上不规则区域的面积。

在实际问题中，经常出现不规则平面如：湖泊、森林、居民区等面积的计算问题。对于这类不规则面积的近似计算，可以沿区域边界，按逆时针(或顺时针)方向取一系列坐标点(x_i, y_i)。然后，计算由这些点构成的多边形的面积 S，并用 S 作为不规则区域的面积的近似值。显然，所取折线愈接近边界曲线，则 S 的精度愈高。

已知：多边形n个顶点的坐标(x_i, y_i)，$i=1, 2, \ldots, n$ (逆时针方向)。

面积计算公式：

$$S=\left|\frac{1}{2}\sum_{i=1}^{n} y_i(x_{i-1}-x_{i+1})\right|$$

其中，$i=1$ 时 x_{i-1} 为最后一点的横坐标值，$i=n$ 时，x_{i-1} 为最初一点的横坐标值。

由于 $i=1$ 及 $i=n$ 时，(x_{i-1}, x_{i+1})点的特殊性，将上式改写为：

$$S=\left|\frac{1}{2}\left[y_1(x_n-x_2)+\sum_{i=2}^{n-1} y_i(x_{i-1}-x_{i+1})+y_n(x_{n-1}-x_1)\right]\right|$$

程序：

```
DIM  x(1), y(1)
INPUT  prompt "How many points?": n
MAT  REDIM  x(n).y(n)
FOR  i=1 to n
     PRINT  "x("; i; "), y("; i; ")=";
     INPUT  x(i), y(i)
LET  s=(y(1)*(x(n)-x(2))+y(n)*(x(n-1)-x(1)))
FOR  i=2 to n-1
     LET  s=s+y(i)*(x(i-1)-x(i+1))
NEXT  i
LET  s=abs(s/2)
PRINT  "S="; s
END
```

例 25　高斯-约当消去法解线性方程组

对于线性方程组：

$$A_{11}X_1+A_{12}X_2+A_{13}X_3+\cdot\cdot\cdot+A_{1n}X_n=B_1$$
$$A_{21}X_1+A_{22}X_2+A_{23}X_3+\cdot\cdot\cdot+A_{2n}X_n=B_2$$
$$\ldots\ldots$$
$$A_{n1}X_1+A_{n2}X_2+A_{n3}X_3+\cdot\cdot\cdot+A_{nn}X_n=B_n$$

可用矩阵表示为：

$$AX=B$$

高斯-约当消元方法，是对 A 的增广矩阵

$$[A \vdots B]$$

作一系列的行初等变换，将 A 化为单位对角矩阵 I，相应的 B 变为 B'：

$$[I \vdots B']$$

于是求得方程组的解：

$$X_i=B'_i,\quad i=1,\ 2,\ \ldots,\ n$$

完成上述变换需要作 n 次归一化和消元。如第 k 次归一化与消元：

(1) 归一化是将增广矩阵 A 的第 k 个主对角元素化为 1。需用 A_{kk} 遍除第 k 行的各元素。即

$$A_{kj}\leftarrow(A_{kj}/A_{kk})$$

这里用 A_{kk} 作除数。当 A_{kk} 很小时，会降低计算精度，而且可能导致溢出（商数太大）。特别是当 $A_{kk}=0$ 时，运算就会中断。为了避免这种情况发生，在作归一化处理前应先在该列中选主元（该列中绝对值最大的元素），并将它换到第 k 行。

(2) 消元　是将增广矩阵 A 第 k 列中的非主对角线元素化为零。即，将第 i 行元素减去第 k 行对应元素与 A_{ik} 之积，则有第 i 行 $A_{ik}=0$。当 i 由 1 变到 n（$i\neq k$），则第 k 列非主对角线元素都为零。

程序：

```
DIM  a(1,1)
READ  n
! 读入并显示增广矩阵A
MAT  READ  a(n, n+1)
MAT  PRINT  a;
! 作n次归一化和消元
FOR  k=1 to n
     LET  c=k
     FOR  i=k+1 to n
          IF  abs(a(i,k))>abs(a(c,k)) then LET c=i
     NEXT  i
     ! 对主元作判断
     IF  abs(a(c, k))<1e-20  then
         PRINT  "No solution"
```

```
    STOP
    ELSE  IF c< >k   then
          FOR  j=k to n+1          ! 将主元所在行与第k行交换
              LET  t=a(c,j)
              LET  a(c,j)=a(k,j)
              LET  a(k,j)=t
          NEXT  j
    END  IF
    ! 归一化
    LET  z=a(k,k)
    FOR  j=1 to n+1
        LET  a(k,j)=a(k,j)/z
    NEXT  j
    ! 消去第k列非对角线元素
    FOR  i=1 to n
        LET  m=a(i, k)
        IF  i< >k then
            FOR  j=1 to n+1
                LET  a(i, j)=a(i, j)-m * a(k, j)
            NEXT  j
        END  IF
    NEXT  i
NEXT  k
! 输出变换后的增广矩阵和方程组的解
MAT  PRINT  a;
FOR  i=1 to n
     PRINT  "x("; i; ")="; a(i,n+1)
NEXT  i
DATA  3
DATA  1, -1, 1, -4
DATA  5, -4, 3, -12
DATA  2, 1, 1, 11
END
```

试算数据的精确解为: $x_1=3$ ， $x_2=6$ ， $x_3=-1$。

***例 26**　曲线的多项式拟合。

在本章例 5 中针对一组实验观测数据作出了线性拟合。如果数据的分布不具有近似线性的性质，则应当用多项式作为未知函数的近似，即多项式拟合。

一般考虑用一个 m 次多项式(一般 $m<6$):

$$Y(x)=A_0+A_1x+A_2x^2+\cdots+A_mx^m=\sum_{j=0}^{m}A_jx^j$$

来拟合 n 个数据点对(x_i,y_i)，$i=1, 2, \ldots, n$。其中， m 远小于 n。

作偏差平方和

$$F(A_0, A_1, A_2, \ldots, A_m)=\sum_{i=1}^{n}(Y(x_i)-y_i)^2$$

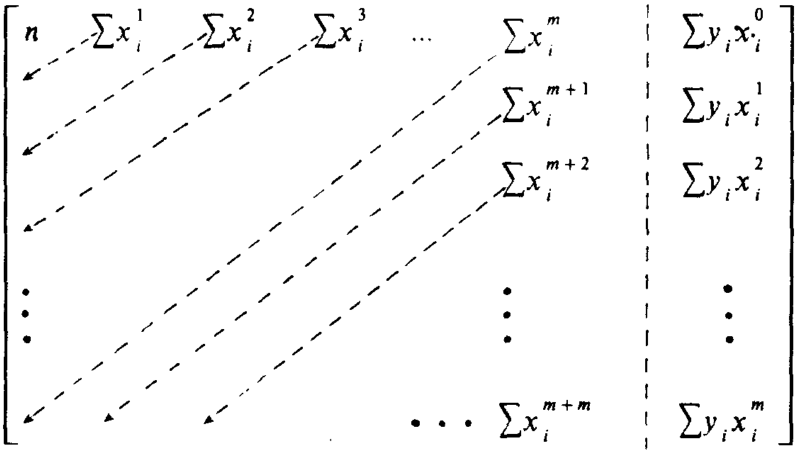
分别对 A_0, A_1, A_2, …, A_m 求偏导数，并令其等于零，得

$$\frac{\partial F}{\partial A_j}=2\sum_{i=1}^{n}(Y(x_i)-y_i)^2 x_i^j=0,\ j=0,\ 1,\ 2,\ \ldots,\ m$$

这个方程组称为最小二乘正规方程组(方程个数与未知数个数相同). 利用本章例 25 或例 19，可以解此线性方程组.

由于此方程组系数矩阵的计算较复杂，下面给出计算这些系数并形成系数增广矩阵的算法过程和相应的程序。

系数的特点：同一反对角线上的元素都相等。因此，只需计算矩阵的第一行和第 $m+1$ 列中的 $2m+1$ 个元素，每算出一个就将它赋给反对角线上的其他元素。然后计算右端常数项中的 $m+1$ 个元素(图 8-1)。其中，求和符∑ 的指标 i 从 1 到 n。



n
$\sum x_i^1$
$\sum x_i^2$
$\sum x_i^3$
…
$\sum x_i^m$
$\sum y_i x_i^0$
$\sum x_i^{m+1}$
$\sum y_i x_i^1$
$\sum x_i^{m+2}$
$\sum y_i x_i^2$
…
$\sum x_i^{m+m}$
$\sum y_i x_i^m$

图 8-1 ·方程组系数矩阵计算

算法过程逻辑框图如图 8-2。其中，循环求$\sum x_i$ 及将它赋给反对角线的过程如图 8-3。

程序:

```
DIM  x(1), y(1), e(0:1, 0:1)
INPUT  n, m                                   ! 输入数据点数 n 和多项式次数 m(m<<n)
MAT  REDIM  x(n), y(n), e(0:m, 0:m+1)         ! E 数组表示系数增广矩阵
DATA  1, 2, 3, 4, 5, 6, 7, 8, 9, 10           ! 数据 x_i 及数据 y_i
DATA  1.987, 4.998, 10.003, 16.994, 25.972
DATA  37.003, 49.582, 65.004, 81.997, 101.01
MAT READ  x, y
LET  e(0, 0)=n
FOR  k=1 to 2*m
     IF  k<m then
         LET  p=0
         LET  q=k
     ELSE
         LET  p=k-m
         LET  q=m
```

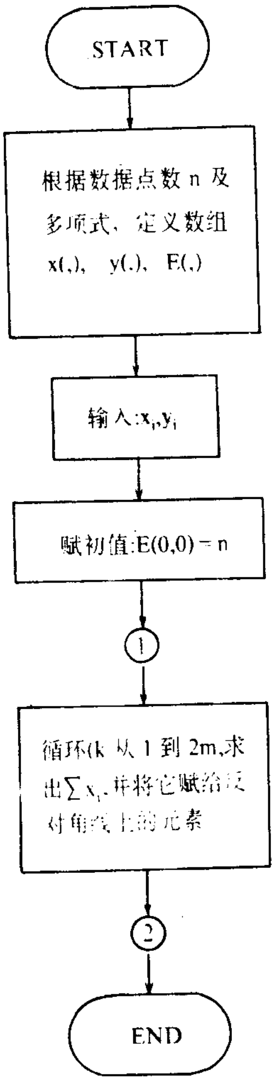

START
根据数据点数 n 及多项式，定义数组 x(,), y(,), E(,)
输入:x_i,y_i
赋初值:E(0,0)=n
①
循环(k 从 1 到 2m,求出$\sum x_i$,并将它赋给反对角线上的元素
②
END

图 8-2　系数矩阵计算逻辑框图

1、2循环图如图 8-3

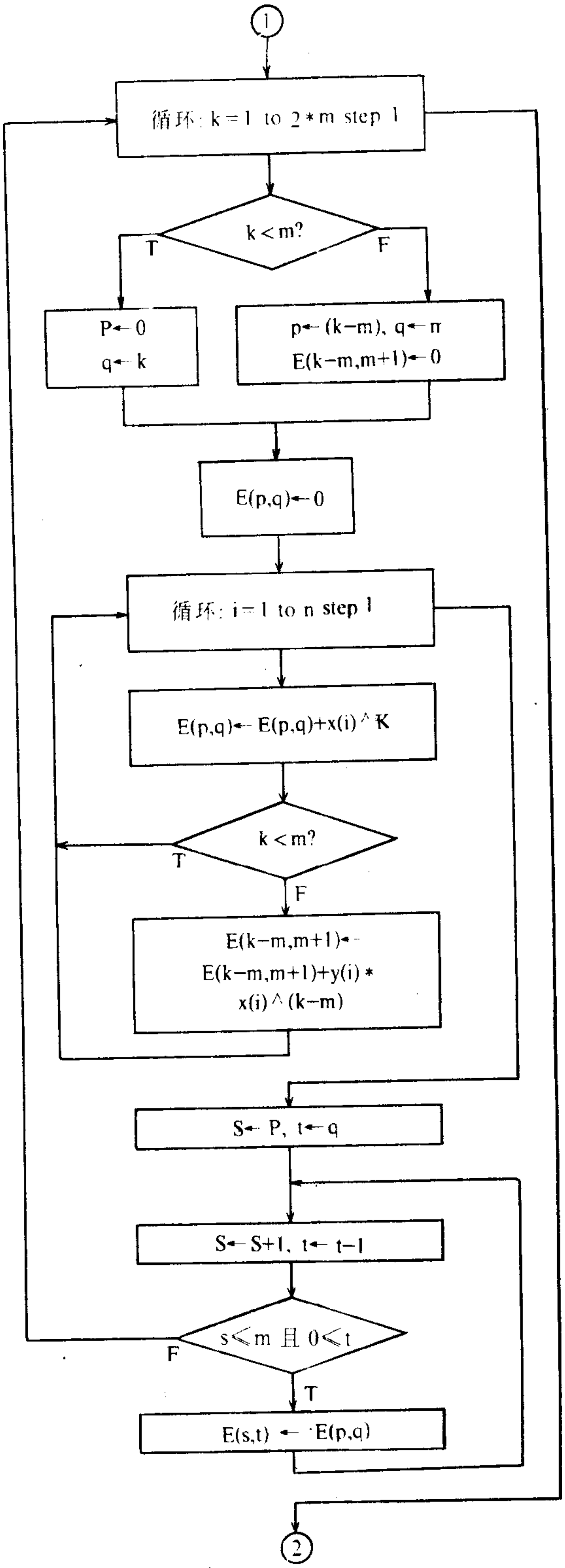

①
循环: k=1 to 2*m step 1
k<m?
T
F
P←0
q←k
p←(k−m), q←m
E(k−m,m+1)←0
E(p,q)←0
循环: i=1 to n step 1
E(p,q)←E(p,q)+x(i)^K
k<m?
T
F
E(k−m,m+1)←E(k−m,m+1)+y(i)*x(i)^(k−m)
S←P, t←q
S←S+1, t←t−1
s≤m 且 0≤t
F
T
E(s,t)←E(p,q)
②

图 8-3　循环求$\sum x_i$及赋给反对角线过程逻辑框图

```
        LET  e(k-m, m+1)=0
    END  IF
    LET  e(p, q) =0
    FOR  i=1 to n
        LET  e(p,q)  =  e(p, q) +x (i) ^k
        IF  not(k<m) then
            LET  e(k-m, m+1)=e(k-m, m+1)+y(i) * x(i)^(k-m)
        END  IF
    NEXT  i
    ! 将求出的元素值赋给反对角线上的各元素
    LET  s=p
    LET  t=q
    DO
        LET  s=s+1
        LET  t=t-1
        IF  (s<=m) and (t>=0) then
            LET  e(s,t)=e(p,q)
        ELSE
            EXIT  DO
        END  IF
    LOOP
NEXT  k
MAT  PRINT  using"#######.###": e
END
---------------------------------------
Ok.RUN
? 10, 2
    10.000       55.000       385.000       394.000
    55.000      385.000      3025.000      3077.025
   385.000     3025.000     25333.000     25697.849
Ok.
```

将求得的系数增广矩阵，代入例 25 求解线性方程组的程序，即可解出 A_0，A_2，…，A_m。从而获得拟合该组实验数据的多项式函数。

习 题 8

(1) 在True BASIC内部，数值型数组的每个元素占用的内存空间都是8个字节，以下定义的四个数组中，占内存空间最大的是(　)，最小的是(　)。

①DIM a(10, 10)　②DIM b(−10: 10)　③DIM C(5, 5, 5)　④DIM D(101)

(2) 随机产生二维数组A的数据，统计A中0的个数及在数组元素中所占的百分比。

(3) 若一个数从左向右读和从右向左读都相同，则称之为回文数，如：686。写程序求出 1000 以内的所有既是回文数同时又是素数的自然数。

(4) 写出下列程序的运行结果:

```
①DIM  A(6)
 DO
     READ  x
     READ  A(x)
     PRINT  A(x)
     LET  n=n+1
 LOOP  UNTIL  n=6
 PRINT
 FOR i=1  to n
      IF MOD(i,2)=0  THEN
          LET  t=A(i)
          LET  A(i)=A(i-1)
          LET  A(i-1)=t
      END  IF
 NEXT  i
 MAT  PRINT  A:
 DATA  1, 2, 3, 1, 4, 3, 2, 4, 5, 6, 6, 5
 END
```

```
②DIM  D(4)
 FOR  i=1  to  3
        READ  D(4-i)
        PRINT  D(4-i)
 NEXT  i
 PRINT
 IF  D(i)<=0  THEN  RESTORE
 FOR  i=1  to   D(2)
        READ D(i)
 NEXT  i
 MAT  PRINT  A;
 DATA  1, 2, 3, 4, 5, 6, 7, 8
 END
```

(5) 找出二维数组中最大元素和最小元素所在的行和列。

(6) 已知一个5×5的数表（数据由DATA语句给出）。请编制程序，对键盘输入的某个数据从表中进行查找，若能找到该数，则显示此数在表中的位置（*m*行，*n*列）以及它与同行同列所有数据的平均数之差；否则，显示“NOT FOUND”。

(7) 求出5×5数组中所有元素的平均值以及大小等于平均值的元素所占的百分比。

(8) 在如下数盘中找出两组4个紧邻数，其和分别为所有4个紧邻数之和中的最大与最小:

┌→11→14→ 3→12→ 5→20→ 1→ 8→ 9→13┐
└ 6←10←15← 2←17← 4←19← 7←16←18←┘

(9) 编号为1到N的N个人按次序围成一圈，从第K个人开始，由1依次报数，报到M者出列，剩余者接着从1报数。如此循环，直到全部出列，求出出列次序。

(10) 在3×3矩阵中，填入数字1～9，使每一行、每一列及对角线上的元素之和都相等。

第 9 章　子程序、函数与库

在程序设计实践中可以发现，许多程序设计课题中往往含有一些可以使用相同的程序设计方法来解决的类似的局部问题。为了提高程序设计效率，通常可以将一些常用的算法编制成独立的程序段。在需要时，这些独立的程序段可以作为通用的程序部件装拼到各种应用程序中去。

这种用于构造程序的程序部件，称为程序的逻辑模块。运用模块构造程序，是结构化设计的重要内容。

True BASIC 所涉及的逻辑模块主要有子程序和函数过程两种形式。

9.1　子程序

能够完成某种确定功能的语句序列，当它作为独立的程序部件以语句块的形式用于主程序中时，便是一个子程序（subroutine）。

子程序一般是为实现某种特定的运算处理而设计的，因而都具有确定的功能。但是，由于子程序是作为逻辑模块，而不是作为完整的程序来设计的，所以子程序与普通程序不同，它不能单独运行，必须由主程序调用。主程序可以根据不同的需要来调用具有相应功能的子程序。主程序调用子程序时，一般先要在主程序与子程序之间进行数据传递。即主程序把与运算处理有关的数据代到子程序中，然后，再由子程序把运算处理的结果送回主程序中（常称为“返回”）。

主程序与子程序之间的关系如图 9-1 所示。

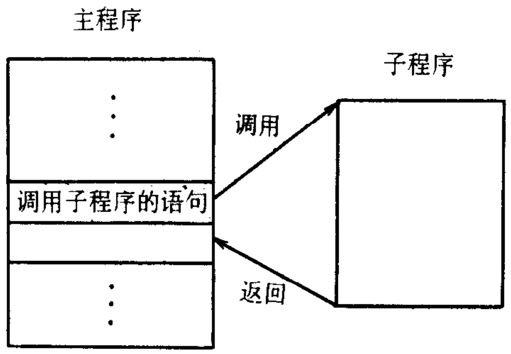


图 9-1　主程序与子程序的关系

9.1.1　定义子程序

每个子程序除了包含完成指定功能的一组语句外，还必须具有子程序名，作为子程序的标识。如果子程序运行时需要从外部输入数据或者向外部输出数据，还应列出用来与外部传递数据的各个变量。子程序的格式如下：

```
SUB  subname(var1,  var2,  …)
     ……
     ……  }  (语句块)
     ……
END  SUB
```

其中，SUB 标记子程序的开始，END　SUB 标记子程序的结束。SUB 和 END　SUB 构成一个子程序模块，子程序所用的各条语句必须置于 SUB 和 END　SUB 之间。subname 为子程序名，可按照变量名的命名规则任意给定。圆括号中的变量称为形式参数，起对外传输数据的作用。若子程序不与外部交换数据，则 SUB 后只写子程序名，不写形式参数表。

例 1　编写一个求任意两个整数的最大公因子的子程序。

取子程序名为 gcd；以变量 m，n 作为输入，接收两个整数；以变量 d 作为输出，返回，以求得最大公因子。

子程序:

```
SUB  gcd(m, n, d)
     FOR  i=min(m,n) to 1 step  -1                ! 从min(m,n)开始向下查找
          IF  mod(m,i)=0 and mod(n,i)=0 then      ! 若同时整除m, n则:
              LET  d=i                            ! 将公因子赋给d
              EXIT  FOR                           ! 退出循环, 结束查找
          END  IF
     NEXT  i
END  SUB
```

子程序中实际使用了 4 个变量 m，n，d 和 i，由于循环变量 i 仅用于子程序当中而与外部无关，因此不必在形式参数表中加以说明。

在子程序中，可以使用 EXIT SUB 语句退出子程序。例如，可以将上例的子程序改为:

```
SUB  gcd(m, n, d)
     FOR  i=min(m,n) to 1 step  -1
          IF  mod(m,i)=0 and mod(n,i)=0 then
              LET  d=i
              EXIT  SUB
          END  IF
     NEXT  i
END  SUB
```

9.1.2 调用子程序

语句格式:

```
CALL  subname(expr1,expr2,…)
```

作用: 调用名字为 subname 的子程序。

括号内的表达式称为实际参数。实际参数可以是变量、常数或者运算表达式。调用子程序时，借助实际参数与形式参数的对应关系来实现主程序与被调用子程序间的数据传递。

为了便于理解主程序与子程序之间如何传递数据，可以认为，对子程序的调用是按以下规则进行的:

(1) CALL 语句中的各个实际参数，按顺序与被调子程序的各个形式参数逐一对应。

(2) 调用子程序时，先将各实际参数的当前值分别传给对应的形式参数，并代入子程序中运行。

(3) 子程序运行结束后，各形式参数的新值将分别送回给主程序中对应的实际参数。此时如果实际参数是一个变量，这个变量就会接收到子程序对应形式参数的返回值（常数或运算表达式的实际参数没有返回值）:

```
CALL  subname(实际参数 1,实际参数 2,…)
            代入↓ ↑返回   代入↓ ↑ 返回
 SUB  subname(形式参数 1, 形式参数 2,…)
```

调用子程序时，实际参数应当与对应形式参数的类型（数值或字符串）和形态（简单变量或数组）一致，且个数相等。实际上，子程序中的形式参数只是“形式上”起着变量的作用，用来说明所传递的数据的类型、形态以及在子程序中的代入位置。在子程序运行时，实

际处理的是与形式参数相对应的实际参数。因此，只要类型和形态正确，形式参数的取名可以是任意的。

为了对子程序进行调用，可以将子程序与主程序拼接在一起。例如，将子程序置于主程序 END 语句的后面。

例 2 利用子程序 gcd 编制一个计算任意两个整数的最大公因子的完整程序。

程序:

```
! 主程序                                        ---------------------------
INPUT  prompt″n1, n2=″: n1, n2                  Ok.RUN
CALL  gcd(n1,n2,d)                              n1, n2=42, 35
PRINT  ″d=″; d                                  d= 7
END                                             Ok.
! 子程序
SUB  gcd(m,n,d)
     FOR  i=min(m,n) to 1 step  -1
          IF  mod(m,i)=0 and mod(n,i)=0 then
             LET  d=i
             EXIT  SUB
          END  IF
     NEXT  i
END  SUB
```

子程序的每一个形式参数既可用于输入数据，也可用于输出数据：如果子程序的运算依赖于开始时某个形式参数所接收的值，那么这个形式参数便起着输入数据的作用；如果子程序运行后某个形式参数取得一个新值，那么新值将回送给对应的实际参数，于是这个形式参数便起着输出数据的作用。

9.1.3 形式参数与实际参数的书写规则

True BASIC 规定：子程序的形式参数必须是简单变量或完整的数组。形式参数的取名方法与变量的类型与形态有关：

(1) 如果形式参数是简单变量，那么直接使用简单变量名。如例 1 所示；

(2) 如果形式参数表示一个完整的数组，那么形式参数应是后接一对圆括号的数组名。当数组是二维以上时，括号内还需用逗号来说明数组的维数。例如 a( )，z(,)，name$(, ,) 表明它们分别是一维、二维的数值数组和三维的字符串数组。

调用子程序时，CALL 语句的实际参数必须与子程序的形式参数在类型、形态和个数上完全对应：

(1) 如果子程序的形式参数是简单变量，则相应的实际参数应当是单值表达式（包括常数、简单变量、下标变量或运算表达式）；

(2) 如果子程序的形式参数是数组，则相应的实际参数也必须是数组，同形式参数一样，用后接一对圆括号并通过逗号标明维数的数组名来表示（在使用 1.0 版本的 True BASIC 时，作为数组的实际参数只用数组名，后面不能带圆括号）。

主程序对于子程序的调用是很灵活的。假设子程序 demo 的形式参数按如下方式定义：

```
SUB  demo(a( ), p$,s)
```

那么对子程序 demo 可以有多种的调用形式：

```
CALL  demo(r( ),name$(i),a )
CALL  demo(q( ),"a",3)
CALL  demo(r( ),b$&"a",x-1 )
```

等等，只要实际参数符合对应形式参数所要求的类型、形态和个数即可。

例 3 编制一个子程序 swap，对二维数值数组的 *i*，*j* 两行进行交换。

对二维数组 a 的 *i*，*j* 两行进行交换，可以通过逐列交换第 *i* 行和第 *j* 行元素的过程来实现。

子程序：

```
SUB  swap(a(,) ,i,j)
      FOR  k=lbound(a,2) to ubound(a,2)        ! k 循环遍历数组 a 的列下标
           ! 对第 k 列上 i，j 两行的元素进行交换
           ! 交换时利用变量 t 暂存数据
           LET  t=a(i,k)
           LET  a(i,k)=a(j, k)
           LET  a(j,k)=t
      NEXT  k
END  SUB
```

调用子程序 swap，便可对二维数组的任意两行进行交换：

```
DIM  d(5,4)
MAT  READ  d
DATA  1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20
MAT  PRINT using"####": d
INPUT  prompt "i, j=?": i, j
CALL  swap(d(,), i,j)
MAT  PRINT using"####": d
END
!
SUB  swap(a(,), i, j)
      FOR  k=lbound(a,2) to ubound(a,2)
           LET  t=a(i,k)
           LET  a(i,k)=a(j, k)
           LET  a(j,k)=t
      NEXT  k
END  SUB
---------------------------
Ok.RUN                                        i, j=? 2, 4
   1    2    3    4                              1    2    3    4
   5    6    7    8                             13   14   15   16
   9   10   11   12                              9   10   11   12
  13   14   15   16                              5    6    7    8
  17   18   19   20                             17   18   19   20
                                              Ok.
```

例 4　编制一个能对任意 $n \times m$ 的二维数值数组按列对行排序的子程序。要求：

(1) 能够按任一列元素的数值来排序，排序时对行的顺序关系进行调整；

(2) 能够指定排序方式："UP"（升序）或"DOWN"（降序）。

作为具体实例，考虑以下成绩表(5×2 数值数组)：

编号	成绩
52	67.5
105	89.0
74	100.0
214	54.5
135	95.0

按第 2 列（成绩）排序

$\xrightarrow{\qquad\qquad}$

排序方式为"DOWN"（降序）

编号	成绩
74	100.0
135	95.0
105	89.0
52	67.5
214	54.5

对二维数组按第 k 列对行排序时，可以将第 k 列看成一个一维数组，并利用一维数组的排序算法对第 k 列进行排序。同时，为了保持行的对应关系，在对第 k 列排序的过程中，如果两元素需要进行交换，则第 i 行和第 j 行必须整行交换。

利用例 3 中的子程序 swap，可编制如下的排序子程序 sort：

```
SUB  sort(a(,), k, p$)          ! k 为排序的列号，p$ 为排序方式
     FOR  i=lbound(a,1) to ubound(a,1)-1
          FOR  j=i+1 to ubound(a,1)
               IF ucase$(p$)="DOWN"  and  a(i,k)<a(j,k) then
                  ! 如果排序方式为降序且第 k 列上第 i 行元素小于第 j 行元素则
                  ! 交换 i, j 两行所有元素
                  CALL  swap(a(,), i,j)
               ELSE IF  ucase$(p$)="UP"  and  a(i, k)>a(j, k) then ! 排升序
                   ! 如果排序方式为升序且第 k 列上第 i 行元素大于第 j 行元素则
                   ! 交换 i, j 两行所有元素
                   CALL  swap(a( , ), i, j)
               END  IF
          NEXT  j
     NEXT  i
END  SUB
```

子程序 sort 中有两处调用了子程序 swap 。

有了子程序 sort 和 swap，为实现排序另编一主程序，并将 sort 和 swap 子程序置于其后，构成一完整程序。

```
DIM  d(5,2)
MAT  READ  d
INPUT  prompt "k=?" : k
INPUT  prompt "up or down?": p$
CALL  sort(d(,), k, p$)
MAT  PRINT d
DATA  52,76.5,105,89,74,100,214,54.5,135,95
END
```

```
!
SUB  swap(a( , ), i, j)
     FOR  k=lbound(a, 2) to ubound(a, 2)
          LET  t=a(i, k)
          LET  a(i, k)=a(j, k)
          LET  a(j, k)=t
     NEXT  k
END  SUB
!
SUB  sort(a(,), k, p$)
     FOR  i=lbound(a,1) to ubound(a,1)-1
          FOR  j=i+1 to ubound(a,1)
               IF  ucase$(p$)="DOWN"  and  a(i,k)<a(j,k) then
                    CALL  swap(a(,), i,j)
               ELSE  IF ucase$(p$)="UP"  and  a(i,k)>a(j,k) then
                    CALL  swap(a( , ), i, j)
               END  IF
          NEXT  j
     NEXT  i
END  SUB
--------------------------------
Ok.RUN                                        k=?  2
 52          76.5                             up or down? down
 105         89                                74          100
 74          100                               135         95
 214         54.5                              105         89
 135         95                                52          76.5
                                               214         54.5
                                              Ok.
```

其中，主程序通过CALL语句调用排序子程序sort时，将数组d，列号k及指定排序方式的字符串p$传给子程序sort。子程序按所给的列号*k*和所指定的方式p$对数组d进行排序，然后将排好序的数组d返回给主程序。

在以上例子中，主程序调用子程序sort，而子程序sort又调用了另一子程序swap。True BASIC允许子程序的多级调用，并且对调用的级数没有限制。利用多级调用子程序的方法，可以层次式地构造程序（图9-2）。

从例4还可以看出：程序设计中引入了子程序方法后，可以把一个大程序拆成几个相对独立的部分，按功能模块分别进行设计。程序中各功能模块之间的联系，由于采取模块调用的形式而变得更加清晰。

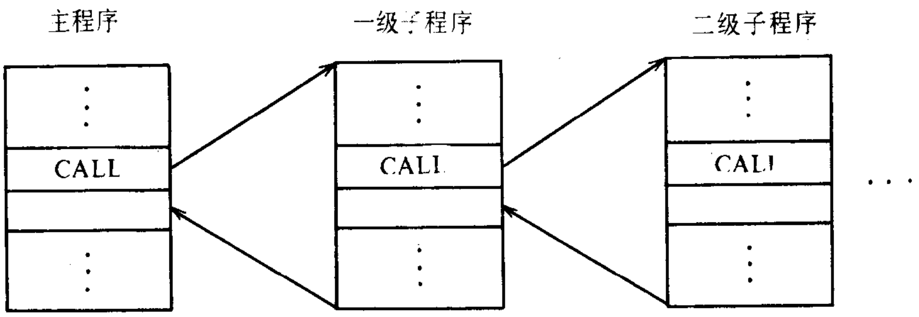


图 9-2　主程序与子程序的层次关系

9.1.4　内部子程序与外部子程序

True　BASIC 的子程序又分为内部子程序和外部子程序两种：位于主程序 END 语句之前（即主程序之内）的子程序为内部子程序；位于主程序 END 语句之后或建立在库中（即主程序之外）的子程序为外部子程序。内部子程序与外部子程序具有相同的格式，区别仅在于所处的位置不同。

许多情况下，一个子程序既可作为内部子程序又可作为外部子程序来使用。在例 2 和例 3 中，如果将子程序调到主程序 END 语句之前的任意位置上，程序的运行结果是相同的。尽管内部子程序位于主程序之内，但是当执行主程序语句序列时，将会绕过子程序。无论是内部子程序还是外部子程序，只有经 CALL 语句调用，才会被执行。

必须注意，内部子程序中的变量与外部子程序中的变量（指实际使用的变量，不包括形式参数）的实际作用有很大差别：内部子程序中的所有变量都是“全局变量”，而外部子程序中所有的变量都是“局部变量”。所谓“全局变量”是指在整个程序中共用的变量；“局部变量”则是指仅在某个局部（比如子程序内部）起作用的变量，它们不能被程序的其它部分所共用。因此，当内部子程序中的某一变量(不包括形式参数)与主程序或其它内部子程序的变量同名时，这些同名变量均指同一个变量。如果在子程序中改变了该变量的值，那么将会影响主程序或其它内部子程序。而外部子程序与主程序或其它子程序之间，仅仅是通过形式参数传递数据，没有变量共用的问题。即便使用了相同的变量名，也是表示不同的变量。显然，无论是内部子程序还是外部子程序，其形式参数总是局部的，因为它们仅在子程序中起作用。

由于内部子程序中的变量是全局的，外部子程序的变量是局部的，因此，对于同一个子程序，分别把它作为内部子程序或外部子程序时，可能会导致不同的运行结果。例如：

程序(1):

```
! 内部子程序
SUB  subr
     LET  i=i+1
     PRINT  i;
END  SUB
! 主程序
FOR  i=1 to 10
     CALL  subr
```

程序(2):

```
! 主程序
FOR  i=1 to 10
     CALL  subr
NEXT  i
END
! 外部子程序
SUB  subr
     LET  i=i+1
```

```
NEXT  i
END
----------------
Ok. RUN
2  4  6  8  10
Ok.
```

```
    PRINT  i;
END  SUB
----------------
Ok. RUN
1 1 1 1 1 1 1 1 1 1
Ok.
```

程序(1)中的subr 为不带形式参数的内部子程序，其中 i 为全局变量，因此与主程序中的循环变量 i 是同一变量。当主程序在循环过程中每次调用 subr 时，由于循环变量 i 在 subr 中增加 1，因而缩短了主程序中原定的循环过程，相当于使原来的步长增加了 1。而程序(2)中的 subr 是外部子程序，其中 i 为局部变量。虽然它与主程序的循环变量 i 同名，但它们是相互独立的两个不同的变量。因此 subr 中使 i 加 1 不会影响主程序的循环过程。由于 subr 不带形式参数，主程序与子程序之间没有数据传递，所以每次调用 subr 时，子程序本身所产生的 i 值均为 1。

需要注意的是：每次调用外部子程序时，总是先将外部子程序中的所有数值变量初始化为零，所有字符串变量初始化为空串，前次调用取得的变量值不被保留。

子程序与主程序往往是分开单独设计的。一个独立设计的子程序作为内部子程序使用时，可能会因为其中的变量与主程序或其它内部子程序的变量同名，造成变量使用上的混乱，从而导致意想不到的错误。为了能够在各自的设计中任意使用变量名，一般情况下子程序应当以外部子程序的方式使用。

例 5 计算并显示 1 到 5 的阶乘数。分别独立设计的子程序和主程序如下：

子程序：

```
SUB  f(n)
     LET  s=1
     FOR  i=1 to n
          LET  s=s*i
     NEXT  i
     PRINT  n; "!="; s
END  SUB
```

主程序：

```
FOR  i=1 to 5
     CALL  f(i)
NEXT  i
END
```

将子程序 f 作为外部子程序来使用时，程序可得到正确结果。

```
! 主程序
FOR  i=1 to 5
     CALL  f(i)
NEXT  i
END
! 外部子程序
SUB  f(n)
     LET  s=1
     FOR  i=1 to n
          LET  s=s*i
     NEXT  i
     PRINT  n;"! =";s
END  SUB
```

```
------------------
Ok. RUN
1 ! = 1
2 ! = 2
3 ! = 6
4 ! = 24
5 ! = 120
Ok.
```

然而，将子程序 f 作为内部子程序来使用时，程序得出的结果却是错误的。

```
! 内部子程序。其中，循环变量 i，形式参数 n 和主程序的 i 实际上是同一变量
SUB  f(n)
LET  s=1
FOR  i=1 to n          ! 等同于FOR  i=1  to i
     LET  s=s*i
NEXT  i                ! 循环结束后i比循环前增加1
PRINT  n;"! =";  s     ! 等同于PRINT  i;  "!=";  s
END  SUB
! 主程序
FOR  i=1 to 5
     CALL  f(i)        ! 从子程序f返回时i的值已增加1
NEXT  i
END
------------------
Ok. RUN
2 ! = 1
4 ! = 6
6 ! = 120
Ok.
```

这是因为，子程序中的 i（全局变量）与主程序中的 i 被当作同一变量，而且传入子程序形式参数 n 的实际参数也是 i，从而造成程序逻辑上的混乱。

使用外部子程序，可以将各个子程序中的变量与其外部隔离，从而确保程序的逻辑模块可以独立设计和自由调用。

9.2 函　数

True BASIC 除了具有系统内部的标准函数外，还允许用户根据需要自己定义各种函数。用户定义的函数也是独立的程序部件，在程序设计中起着逻辑模块的作用。

作为程序部件，以返回运算值的形式被主程序调用的语句序列，便是一个函数过程。由于用户定义的各种函数均由函数过程来实现，并能象系统内部函数一样使用，因此函数过程也可简称为函数（FUNCTION）。

函数可以具有任意数目和任意类型的自变量。函数值既可以是数值型，也可以是字符型。经过定义的函数，能够象标准函数一样出现在主程序的表达式中。实际上主程序中每进行一次函数计算，便调用运行一次对应的函数模块。

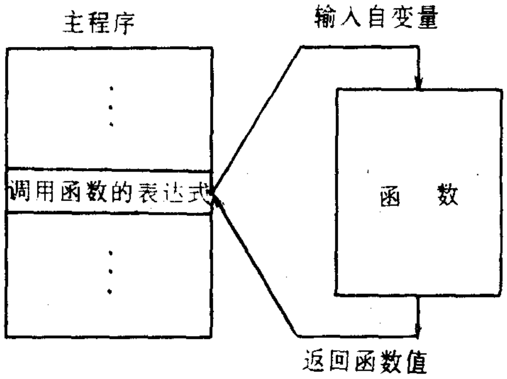


图 9-3　主程序与函数模块的关系

进行函数调用的主程序与对应函数模块的关系如图 9-3 所示。

9.2.1　函数的定义

每个函数除了包含能完成指定函数运算功能的一组语句外，还必须具有函数名和自变量表。

函数的格式如下：

```
DEF  func(var1、var2，…)
     ……
     ……                 ⎫
     LET func = expr    ⎬ (实现函数运算的语句块)
     ……                 ⎭
     ……
END  DEF
```

其中，DEF 标记函数定义的开始，END DEF 标记函数定义的结束。由 DEF 和 END DEF 构成一个函数模块结构，函数运算过程所用的语句必须置于 DEF 和 END DEF 之间。func 为函数名，可以按变量的命名规则任意给定。括号内为函数的各个自变量。自变量为形式参数，应当出现在实现函数运算的有关语句中。与自变量取值对应的函数值定义为向函数名赋值的表达式 expr 的运算结果。

例如，阶乘函数可以定义为：

```
DEF  fac(n)
     LET  a=1
     FOR  i=1 to n
          LET  a=a*i
     NEXT  i
     LET  fac=a        ！将数值 a 赋给函数名 fac 作为函数值
END  DEF
```

又例如，求一维数组前 n 号元素的最大值的函数可以定义为：

```
DEF. big(a( ),n)
     LET  k=lbound(a)
     LET  m=a(k)
     FOR  i=k+1 to k+n-1
          LET  m=max(m,a(i))
     NEXT  i
     LET  big=m        ！将数值 m 赋给函数名 big 作为函数值
END  DEF
```

定义函数时，函数的自变量实际上是形式参数，用于说明自变量数据的类型、形态及在函数运算过程中的位置。自变量的实际数据由每次使用函数时所代入的具体内容来决定。定义函数时，必须在函数运算过程中为函数名赋值，这样才能在调用该函数时获得函数值。函数中可以使用 EXIT DEF 语句退出该函数模块，但应在执行 EXIT DEF 语句退出前，先对函数名赋值。

程序中，函数的定义应当位于所有使用这一函数的语句之前。经过定义以后的函数能够象标准函数一样用于程序的表达式中。调用函数时，代入的自变量按实际参数的规则书写。

例 6　利用自定义函数 big 求一维数组 r 中所有元素的最大值。

程序:

```
DEF  big(a( ),n)
       LET  k=lbound(a)
       LET  m=a(k)
       FOR  i=k+1 to k+n-1
              LET  m=max(m,a(i))
       NEXT  i
       LET  big=m
END  DEF
!
DIM  r(10)
MAT  READ  r
PRINT  big(r( ),size(r))
DATA  12,-24,45,95,-68,32,80,-97,51,0
END
------------------------------
Ok.RUN
 95
Ok.
```

例 7 定义一个字符串函数，从完整的姓名中提取出姓氏。由于英文姓氏是写在名的后面的，因此提取姓氏也就是提取出姓名右侧的第一个单词。

程序:

```
DEF  surname$(name$)
       FOR  i=len(name$) to 1 step -1              ! 对姓名从右到左找空格
              IF  name$[i: i]=" " then EXIT FOR  ! 找到空格则退出
       NEXT  i
       LET  surname$ =name$[i+1: len(name$)]  ! 截取空格右侧的字符串
END  DEF
!
INPUT  prompt "Name=?": n$
PRINT  "Surname="; surname$(n$)
END
---------------------------------------
Ok. RUN
Name=? Robert Green
Surname = Green
Ok.
```

函数 surname$ 中通过 i 循环从右到左在输入的姓名字符串中寻找第一个空格。如果找到空格，则立即退出循环，此时 i 为从右端起的第一个空格的位置；如果找不到空格，则继续循环直至循环结束，此时 i 为-1。因此，无论姓名字符串中是否存在空格，i+1 均为右侧第一个单词的起始位置（结束位置则为整个姓名的最后一位，即 len(name$)）。由于函数

值是字符串，因此函数名必须以 $ 结尾。

函数通过自变量输入数据，利用函数名输出求得的函数值。调用函数时不会改变作为自变量的实际参数的内容，而是把实际参数的值拷贝给相应的形式参数。函数运算时只对形式参数进行操作。这一点与子程序调用是不同的。

以上讨论了自定义函数的一般格式。此外，True BASIC 还提供了一种简化的单行自定义函数格式，适用于定义那些能够表示成单一表达式的函数。

单行函数的格式如下：

```
DEF  func(var1,var2,…)=expr
```

其中，DEF 标记函数定义的开始。当在同一行中含有“=”时，说明本行是一个单行的函数模块。 func 为函数名，可以按变量的命名规则任意定名。圆括号内为函数的自变量。expr 为一表达式，自变量应当出现在 expr 中。与自变量值相对应的函数值则定义为表达式 expr 的运算结果。例如：

```
DEF  s(x,y,z)=sqr(x*x+y*y+z*z)
DEF  max3(a,b,c)=max(max(a,b),c)
```

分别定义了函数名为 s 和 max3 的单行函数。其中函数 s 为三个数平方和的开平方，函数 max3 则在三个数当中取最大值。因此，s 和 max3 可以作为合法函数在主程序的表达式中出现。在定义了函数 s 和 max3 之后，赋值语句 LET n=s(0,6,8)+1 将会把 s(0,6,8)+1 的运算结果 11 赋给变量 n。而输出语句 PRINT max3(2,7,4)会显示出最大值 7。

例 8　已知 a，b 及 $f(x)$，用梯形法求定积分

$$\int_a^b f(x)\mathrm{d}x$$

其中，$f(x)=\sqrt{x}\sin x+x$

若将积分区间 $[a, b]$ 分为 n 等分，并设第 i 段的端点为 x_{i-1}，x_i，长度为 h，则由梯形公式所得的定积分的近似值为

$$\frac{h}{2}\sum_{i=1}^{n}[f(x_{i-1})+f(x_i)]$$

程序：

```
DEF  f(x)=sqr(x)*sin(x)+x        ! 自定义的单行函数。
INPUT  prompt "A, B, N=": a, b, n
LET  h=(b-a)/n
FOR  i=1 to n
     LET  x0=a+(i-1)*h
     LET  x1=a+i*h
     LET  s=s+f(x0)+f(x1)
NEXT  i
LET  t=s*h/2
PRINT  t
END
```

程序首先将被积函数定义成 $f(x)$ 的形式，然后对 $f(x)$ 求定积分。实际上，不同被积函数近似积分的计算过程都是一样的，因此在这个程序中，只需改变 $f(x)$所定义的函数内容，就

可以求出各种被积函数的定积分近似值。

9.2.2 内部函数与外部函数

与子程序一样，函数也根据它与主程序的位置关系分为内部函数与外部函数。将函数模块置于主程序内时，称为内部函数。将函数模块置于主程序后或建立在库中时，则称为外部函数。一个函数既可作为内部函数，也可作为外部函数来使用。不论是内部还是外部函数，函数模块中函数的自变量（即形式参数）总是局部变量。除此之外，内部函数的所有其它变量均为全局变量，外部函数的所有其它变量均为局部变量。

一般情况下，函数的定义应当先于函数的调用，如例 1，例 2，例 3 所示。如果一个函数在函数模块定义之前调用，那么必须用 DECLARE DEF 语句加以声明。

语句格式：

```
DECLARE  DEF  func1,func2,…
```

作用：声明该程序中将要出现的函数 func1， func2、… 是先使用后定义的函数。

由于外部函数总是位于主程序之后，因此所有外部函数都必须用 DECLARE DEF 语句加以声明。 DECLARE DEF 语句一般位于程序的前部。如例 6 的程序可以改写成使用外部函数的形式：

```
DECLARE DEF  surname$               ! 声明 surname$ 是函数
INPUT  prompt "Name=?": n$
PRINT  "Surname=": surname$(n$)
END
!
DEF  surname$(name$)
     FOR  i=len(name$) to 1 step -1
          IF  name$[i: i]=" " then EXIT FOR
     NEXT  i
     LET  surname$ =name$[i+1: len(name$)]
END  DEF
```

由于函数的调用与数组元素的使用在形式上是相同的（比如 h(3)既可能是指以 3 为自变量的函数 h 的函数值，也可能指是数组 h 的下标为 3 的元素），因此如果调用了未经 DEF 语句定义或 DECLARE DEF 语句说明的函数，True BASIC 将由于无法判定是函数还是数组元素而给出出错信息：

```
This must first appear in a DIM or DECLARE DEF
(必须预先在 DIM 或 DECLARE DEF 中出现)
```

与外部子程序相似，外部函数也可以作为一个通用的程序逻辑模块。由于外部函数中的变量都是局部变量，不会与主程序中的变量发生混淆，因此外部函数可以单独设计。

例 9 用牛顿迭代法求三次方程 $ax^3+bx^2+cx+d=0$ 的一个实根。

求方程 $f(x)=0$ 的一个实根的牛顿迭代公式为：

$$x_{n+1}=x_n-\frac{f(x_n)}{f'(x_n)},\ (n=1,\ 2,\ 3,\ \dots)$$

当 $|x_{n+1}-x_n|$ 小于给定误差 e 时，得近似根 $x \approx x_{n+1}$。由代数理论知，奇次多项式方程至少存

在 1 个实根（对于无实根的方程，如 $x^2+1=0$，循环迭代过程将无法终止）。此例中

$$f(x)=ax^3+bx^2+cx+d$$
$$f'(x)=3ax^2+2bx+c$$

令 e =1E-10，定义三次方程的求根函数 RootF 如下：

```
DEF  RootF(a,b,c,d)
    DEF  f(x)=a*x*x*x+b*x*x+c*x+d              ！定义三次多项式
    DEF  df(x)=3*a*x*x+2*b*x+c                  ！定义相应的导数
    LET  x1=1                                   ！设置迭代初值
    DO
        LET  x0=x1
        LET  x1=x0-f(x0)/df(x0)                 ！循环迭代
    LOOP  until abs(x1-x0)<1e-10                ！满足误差要求则退出循环
    LET  RootF=x1                               ！将函数值赋给函数名
END  DEF
```

该函数有 4 个自变量 *a*，*b*，*c*，*d*，分别对应于三次方程的 4 个系数。调用函数时，只要给出这 4 个系数的具体数值，便可求出相应方程的一个实根。函数内部又使用了两个自定义函数 f 和 df，用于简化迭代式的表达。f 和 df 是 RootF 的内部函数，与 RootF 共用 *a*，*b*，*c*，*d* 这 4 个变量。为了利用函数 RootF 求解三次方程的根，可编一个程序对它进行调用，并将 RootF 函数模块置于程序之后，作为外部函数使用。

```
DECLARE  DEF RootF
INPUT  prompt "a, b, c, d=?": a, b, c, d
PRINT  RootF(a,b,c,d)
END
!
DEF  RootF(a,b,c,d)
    DEF  f(x)=a*x*x*x+b*x*x+c*x+d               ！定义三次多项式
    DEF  df(x)=3*a*x*x+2*b*x+c                  ！定义相应的导数
    LET  x1=1                                   ！设置迭代初值
    DO
        LET  x0=x1
        LET  x1=x0-f(x0)/df(x0)                 ！循环迭代
    LOOP  until  abs(x1-x0)<1e-10               ！满足误差要求则退出循环
    LET  RootF=x1                               ！将函数值赋给函数名
END  DEF
-----------------------------------------
Ok. RUN
a, b, c, d=? 2, 6, -1, 3
-3.29049
Ok. RUN
a, b, c, d=? 1, 0, 0, -4
 1.5874
```

```
Ok. RUN
a, b, c, d=? 0, 1, 0, 9
```

(由于方程 $x^2+9=0$ 无实根，程序出现死循环，需按 [Ctrl]+[Break] 退出)。

通过例 4 可以看到，模块可以嵌套，即在一个子程序模块或函数模块中能够置入其它模块。这时，置入其中的模块便是它的内部模块，与它共用除了各自的形式参数以外的所有变量，并且这些内部模块仅为它所用。因此，如果某个子程序或函数要被多个独立的程序部件所调用，那么它就必须作为独立的外部模块来定义。模块的嵌套不能超过两层（由于主程序也算一层，因此内部子程序或内部函数的内部不能再实行嵌套）。对于比较复杂的程序设计，采用模块结构时，一般是以调用外部模块的方式，而不以嵌套内部模块的方式来组织程序。

9.3 库

在前面运用子程序和函数的例子中，无论是内部的还是外部的子程序和函数，总是与主程序装拼在一起，构成一个完整的程序。但是，在许多情况下，子程序和函数是作为通用模块来设计的，它们应当能被各种不同的程序所调用。如果把这些通用的子程序和函数重复地写进每个需要对其进行调用的程序中，则会造成程序设计过程和程序语句内容的重复。

True BASIC 系统具有库的功能。库，是存贮在外存中的一种特殊的程序(因而库是一种文件)。多个通用的“过程”——能完成一定功能的有名字的程序段统称为“过程”，子程序、函数及第 10 章将讨论的图画都是“过程”——可以组合成一个独立的库。库中的子程序和函数都是公共的，它们能以外部子程序和外部函数的形式，提供给库外其它程序调用。任何程序都可以直接调用库中的子程序和函数，而不必将其拼入程序中。这不仅有利于避免重复，而且使得利用逻辑模块来构造程序的设计方法更加灵活、简洁。

9.3.1 库的建立

库不包含主程序，仅由“过程”构成(图 9-4)。每个库中的第一个语句必须是 EXTERNAL（外部的），用以声明库中的过程都是公共模块。在 EXTERNAL 语句之后放置若干个子程序或函数，然后存贮在外存中，便形成一个库。

注意：库不能够以 END 语句结束。

库不同于一般的程序，它本身不能独立运行。库的作用是为其他程序提供逻辑模块。子程序和函数一旦置入库中，就可以成为公用的程序部件，因而更为灵活通用。例如，可以利用本章例 3 和例 4 中的子程序 swap，sort，以及 9.2.1 中的函数 big 共同建立一个文件名为 LIB.TRU 的库：

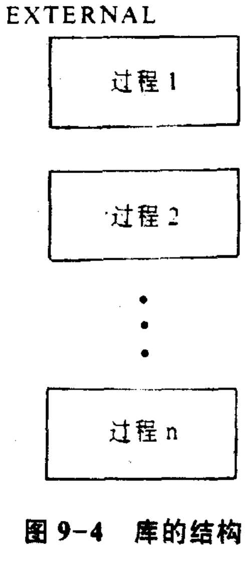


图 9-4 库的结构

```
EXTERNAL
!
SUB  swap(a(,) ,i,j)
     FOR  k=lbound(a,2) to ubound(a,2)
          LET  t=a(i,k)
          LET  a(i,k)=a(j,k)
```

```
        LET  a(j,k)=t
    NEXT  k
END  SUB
!
SUB  sort(a(,), k, p$)
    FOR  i=lbound(a,1) to ubound(a,1)-1
        FOR  j=i+1 to ubound(a,1)
            IF  ucase$(p$)="DOWN" and a(i,k)<a(j,k)  then
                CALL  swap(a(,),i,j)
            ELSE IF ucase$(p$)="UP"  and  a(i,k)>a(j,k)  then
                CALL  swap(a(,),i,j)
            END  IF
        NEXT  j
    NEXT  i
END  SUB
!
DEF  big(a( ),n)
    LET  k=lbound(a)
    LET  m=a(k)
    FOR  i=k+1 to k+n-1
        LET  m=max(m,a(i))
    NEXT  i
    LET  big=m
END  DEF
-----------------------------------------
Ok. SAVE LIB.TRU
```

一般来说，总是将通用子程序和函数组合成适用于解决某类问题的库，即建库遵循按模块功能性质分类的原则。

9.3.2 库的使用

建立库之后，便允许任何程序对库中的子程序和函数进行直接调用。当某个程序需要调用库中的子程序或函数时，必须在程序前部使用 LIBRARY 语句指定所用库的文件名。

语句格式：

LIBRARY “库文件名”

作用：指定所用库的文件名，以便从库中调用子程序或函数。

库中的子程序或函数都是外部的，调用时不会与主程序的变量混淆。需要注意的是：在调用库中的函数之前，应当如同使用外部函数那样用 DECLARE DEF 语句对函数加以说明。例如，建立了库 LIB.TRU 之后，对二维数值数组的排序便可直接调用其中的子程序 sort：

```
LIBRARY "LIB.TRU"
DIM  x(5, 2)
MAT  READ  x
```

```
CALL sort(x, 2,"UP")
MAT PRINT  x;
DATA 1, 36, 2, -7.5, 3
DATA 8.3, 4, 19.8, 5, -2
END
```

```
------------------------
Ok. RUN
 2  -7.5
 5  -2
 3   8.3
 4   19.8
 1   36
Ok.
```

求一维数值数组中的最大元素，也可直接调用其中的函数 big：

```
LIBRARY "LIB.TRU"
DECLARE  DEF  big
DIM  a(5)
MAT  READ a
PRINT  "The biggest is ";  big(a( ), size(a))
DATA  -23, 0, 7.5, -2, 1
END
```

```
------------------
Ok. RUN
The biggest is 7.5
Ok.
```

9.4 递 归

在 True BASIC 系统中，一个子程序或函数不仅可以调用其它子程序和函数，而且还允许子程序和函数调用自身。子程序或函数对自身进行调用的过程称为“递归”。能在程序设计中使用递归技术，是 True BASIC 系统的主要优点之一。

进行程序设计，实际上是利用各种语句对算法进行描述。在前面所有的程序设计例子中，都是将算法的具体过程详细展开，然后按过程的顺序一步步地用相应的语句来表达。这种依照顺序的表达方式，称为演绎方式（或递推方式）。除了演绎方式外，还可以采用归纳方式来描述算法。比如，阶乘的算法可以表示为

$$\begin{cases}1! = 1\\ n! = n \times (n-1)!\end{cases}$$

其中并未说明 $n!$计算过程的每个具体步骤，只给出了初始条件 $1! = 1$ 以及 $n!$ 与前一步 $(n-1)!$ 的关系。利用这个关系，计算 $n!$ 的值可以化归为先计算 $(n-1)!$ 的值，而计算 $(n-1)!$ 的值又可以化归为计算$(n-2)!$的值，……，如此反复回代，直至遇到初始条件 $1! = 1$ 为止。这种逐步化归并最后达到初始条件的计算过程，便是自身调用的过程，即“递归”。逐步化归的关系则称为“递归关系”。实际上，递归体现了数学归纳原理。

实现递归须具备两点：（1）在有限步内可达到的初始条件；（2）逐步趋向初始条件的递归关系。

使用递归往往可以使算法的表达和问题的求解变得自然而简明。

例 10 采用递归的方法定义阶乘函数。

程序：

```
DECLARE  DEF  fac
INPUT  prompt"n=?":n
PRINT  n;"!=";fac(n)
END
```

```
!                                                   -------------
DEF  fac(n)                                         Ok.RUN
     IF  n = 1 then                                 n = ? 4
         LET  fac = 1            ! 初始条件          4 ! = 24
     ELSE                                           Ok.
         LET  fac = n * fac(n-1)  ! 递归关系
     END  IF
END  DEF
```

注意，当所定义的函数被递归调用时（例如，该函数出现在函数模块内某个赋值语句的右侧），必须写出递归调用时的自变量。显然，自变量的取值必须趋向初始条件给出的值。

如果没有初始条件，或者初始条件是不可到达的，那么将导致递归过程无法结束。同样，如果递归关系不是趋向初始条件，也将引起无限递归，而无法得到结果。

例 11　用递归的方法求裴波纳奇数列中的第 n 项。

该数列的递归定义式为：

$$Fb(n)=\begin{cases}1, & (0<n\leqslant 2)\\ Fb(n-2)+Fb(n-1), & (2<n)\end{cases}$$

程序：

```
DECLARE  DEF  fb
INPUT  prompt "n = ?":n
PRINT  "Fb("; n; ") = "; fb(n)
END
!
DEF  fb(n)
     IF  n < = 2 then
         LET  fb = 1      ! 初始条件          -------------
     ELSE                                    Ok.RUN
         LET  fb = fb(n-2)+fb(n-1) ! 递归关系  n = ? 6
     END  IF                                 Fb(6) = 8
END  DEF                                     Ok.
```

例 12　用递归方法求正整数 $n1$，$n2$ 的最大公约数。求最大公约数的递归式定义如下：

$$\gcd(m,n)=\begin{cases}n, & (\text{n能整除m})\\ \gcd(n,\ r), & (n\text{不能整除}m，r\text{为}n\text{除}m\text{的余数})\end{cases}$$

程序：

```
INPUT  prompt "n1, n2 = ?":n1, n2
CALL  gcd(n1,n2,d)
PRINT  d
END
! 求最大公约数 d 的递归子程序
SUB  gcd(m,n,d)
     LET  r = mod(m,n)
```

```
        IF  r=0 then
            LET  d=n
        ELSE
            CALL  gcd(n,r,d)
        END  IF
    END  SUB
```

使用递归算法编程时应当注意：每进行一次递归，计算机都自动保存递归前的状态和数据，以便递归返回后使用。与递推算法相比，使用递归算法往往需要占用较大的内存空间，而且递归的层次越深，对内存的需求越大。所以应当避免过深的递归，否则可能因超出计算机的内存空间而引起出错。

例 13　汉诺(Hanoi)塔问题。

汉诺塔问题源自印度梵教利用搬动汉诺塔来推算宇宙末日的古老方法：设有三根圆柱 a，b，c，有 *n* 张直径各不相同的圆片上小下大地套在柱 a 上，并按下面规则移动圆片：

(1) 搬动过程可使用柱 c;

(2) 每次只将某柱最上面的一张圆片搬到另一柱上;

(3) 每根柱上的圆片都必须按上小下大的要求叠放。

将柱 a 上的圆片全部搬到柱 b 上的过程可递归地表示成:

初始条件——如果 $n=1$，则将 a 上的圆片搬到 b 上。

递归关系——否则做如下 3 步(图 9-5):

(1) 将柱 a 上的 $n-1$ 张圆片按规则搬到柱 c;

(2) 将柱 a 上的第 *n* 张圆片搬到柱 b;

(3) 将柱 c 上的 $n-1$ 张圆片按规则搬到柱 b。

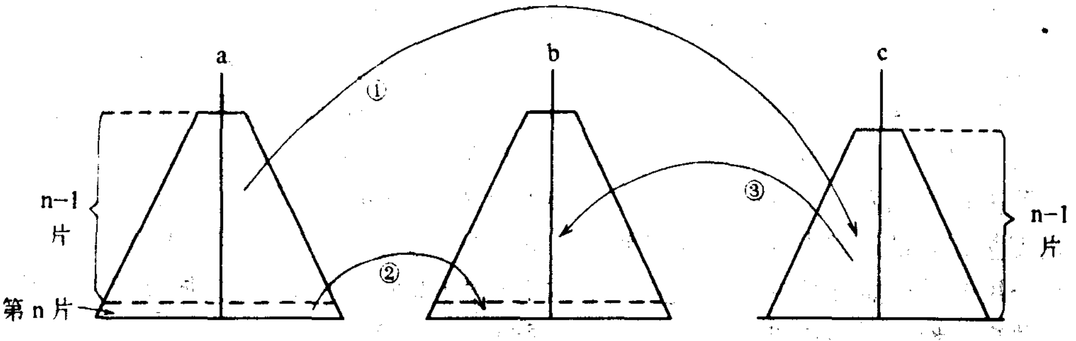


图 9-5　汉诺塔的搬动过程

于是，可以设计一个带 4 个参数的子程序 move(n，x$，y$，z$) 来模拟圆片的搬动。其中，第 1 个参数对应于搬动的圆片数，第 2,3,4 个参数按顺序分别对应于搬动时的源柱，目的柱，辅助柱的柱名。

程序:

```
    INPUT  prompt ″How many pieces?″:h            ! 输入搬动的圆片数
    CALL  move(h, ″a″, ″b″, ″c″)                   ! a、b、c分别为源、目的、辅助柱
    END
```

```
!
SUB  move(n, xS, y$, zS)                   ! 汉诺塔搬动过程的递归子程序
     IF  n=1 then                          ! 如果只有一片则
         PRINT  xS; "---->"; yS            ! 从源柱搬到目的柱
     ELSE                                  ! 否则
         CALL  move(n-1, x$, z$, y$)       ! 将n-1片从源柱搬到辅助柱
         PRINT  x$; "---->"; y$            ! 将第n片从源柱搬到目的柱
         CALL  move(n-1, z$, y$, x$)       ! 再将辅助柱上的n-1片搬到目的柱
     END  IF
END  SUB
```

子程序 move 的后 3 个形式参数按位置分别表示搬动时的源柱、目的柱、辅助柱。搬动 *n* 片时的源柱、目的柱、辅助柱分别是 x$,yS,zS。但在搬动上部的 *n*−1 片时，x$，y$，zS 的作用则与搬动 *n* 片时不同。因此在递归地调用 move 来搬动 *n*−1 片时，这 3 根柱子要根据其作用代入到新的实际参数位置上。

*9.5 公用变量、共享变量和模块

运用外部过程(外部子程序或外部函数)构造程序，是进行结构化程序设计的基本手段。外部过程之所以具有独立性，能够在不同的调用环境下正确运行，是因为它的内部数据与外界隔离，变量是局部的。除了以规定的形式参数与外界传递数据外，其它数据可以被安全地隐蔽在过程内部。对外界环境来说，它们是“透明”的，不可见的，如同不存在一样。

外部过程(除形式参数外)全部使用局部变量，有时也会出现一些问题。比如：(1)如果与外界传递很多数据，势必需要设置很多形式参数，而过多的形式参数不仅不便于书写和阅读程序，而且会增加程序出错的可能性和调试的难度。(2) 同一过程的两次调用之间(或互不调用的两个过程之间)无法传递数据，必须通过形式参数与主程序的某一变量对应，以便前一调用(或某一方)运行结束后，后一调用（或另一方）能够从这个变量中得到运行结果。这样，本来与主程序无关的数据不得不借助主程序来传递，因而破坏了外部过程的独立性。

为了解决这些问题，True BASIC 在 2.0 以上版本中除在主程序(包括所有内部过程)中使用全局变量和外部过程中使用局部变量外，还为外部过程新设了两种形式的变量：

(1) 公用变量：能够被主程序(包括所有内部过程)和所有外部过程共同使用的变量。

(2) 共享变量：能够被相关的外部过程共同使用的变量。

定义公用变量或共享变量，需要将外部过程组装成“模块”。应当注意的是：本节讨论的“模块”是 True BASIC 的一项语法内容，而非程序设计理论中所称的逻辑模块。

9.5.1 模块的定义

模块是外部过程的集合。模块的作用是：将彼此相关的外部过程组合在一起，并确定在这些外部过程中，哪些变量是公用变量，哪些变量是共享变量；并且，模块还能够对公用变量和共享变量进行初始化，使这些变量在使用前具有适当的初值；如果需要，还能够将模块内的某些过程定义成私有过程。私有过程隐蔽在模块内，只能被模块内的其它过程调用。从模块外部不能访问私有过程。

模块的定义格式:

```
MODULE  moduleName
        PUBLIC  语句部分
        ……
        SHARE  语句部分
        ……
        PRIVATE  语句部分
        ……
        模块初始化部分
        ……
        外部过程定义部分
        ……
END  MODULE
```

模块的定义由 MODULE 开始，END MODULE 结束。moduleName 为模块名，可以按变量名的规则任意给定。模块内部一般包括 5 部分。

(1) PUBLIC 语句部分：由一个或多个 PUBLIC 语句组成。PUBLIC 语句用于定义公用变量。凡是在 PUBLIC 语句中说明的简单变量和数组均为公用变量，整个程序的任何部分(包括主程序和所有外部过程)均可共同使用。用 PUBLIC 语句说明数组时必须给出数组各维的上下界。例如:

```
PUBLIC  a,b,c$,d(10,10),e$(-100:100)
```

(2) SHARE 语句部分：由一个或多个 SHARE 语句组成。SHARE 语句用于定义共享变量。凡是在 SHARE 语句中说明的变量和数组均为共享变量，模块内的任何部分均可共享，但对模块外部来说是不可用的。共享变量的值一直在内存中保持，而不是象局部变量那样，外部过程运行结束后其值便会丢失。共享变量因而又称为“静态”变量。用 SHARE 语句说明数组时必须给出数组各维的上下界。例如:

```
SHARE  x, y$, z(10)
```

(3) PRIVATE 语句部分：由一个或多个 PRIVATE 语句组成。PRIVATE 语句用于将模块内的过程定义成私有过程。私有过程的子程序名或函数名应在 PRIVATE 语句中给出。模块内的函数经 PRIVATE 语句说明之后，当被本模块的其它过程调用时不需再用 DECLARE DEF 进行说明。

(4) 模块初始化部分：在 PUBLIC、SHARE 和 PRIVATE 语句之后，外部过程定义部分之前的一段程序。这段程序用于计算公用变量和共享变量的初值。

(5) 外部过程定义部分：列出模块包含的所有外部过程。

构成模块的这 5 个部分都是可选的。一个模块可以同时具有这 5 个部分，也可以省略其中某几部分。

例 14 某质点不断地在平面上移动，计算质点在平面上从一个点到另一个点的移动距离(仅提供出发点的坐标和当前的坐标)。为此，定义 3 个相关的外部过程：子程序 StartFrom(x,y) 记录起点坐标，子程序 MoveTo(x,y) 记录每次移动后的坐标，不带自变量的函数 DISTANCE 则返回最近一次移动的距离。

用变量 x_0，y_0 和 x_1，y_1 表示最近一次移动的前后坐标值。由于 3 个外部过程共同使用这些变量，因此编程时需要定义模块，将这些变量作为模块的共享变量，并将 3 个外部过程置于模块之内。

模块 move:

```
MODULE  move                                 ! 模块 move 开始
     SHARE  x0,y0,x1,y1                      ! 定义共享变量
     !
     SUB  StartFrom(x,y)                     ! 定义子程序 Start From
          LET  x0,x1 = x                     ! 作为起点时没有前一点，
          LET  y0,y1 = y                     ! 因此可令前后坐标一致。
     END  SUB
     !
     SUB  MoveTo(x,y)                        ! 定义子程序 Move To
          LET  x0 = x1                       ! 每移动一次，原来的移动
          LET  y0 = y1                       ! 后坐标变成移动前坐标；
          LET  x1 = x                        ! 重新记录移动后的坐标。
          LET  y1 = y
     END  SUB
     !
     DEF  distance = sqr((x1-x0)^2+(y1-y0)^2) ! 定义函数 distance
END MODULE                                   ! 模块 move 结束
```

9.5.2 模块的使用

模块可以置于主程序之后，也可以置于库中。模块本身并不是一个可调用执行的程序部件。实际上，模块提供给外部使用的，是定义在模块内的各个外部过程。当需要使用这些过程时，可直接进行调用。调用定义在模块内的过程与不使用模块时的调用方法完全相同。

例 15 利用例 14 定义的模块 move 计算平面上从一个点到另一个点的移动距离。

程序:

```
DECLARE  DEF  distance              ! 声明模块中的函数distance为外部函数
CALL  Start From(0,1)               ! 起点为(0、1)
CALL  MoveTo(2,3)                   ! 移动到(2、3)
PRINT  distance                     ! 显示移动距离
CALL  MoveTo(7,8)                   ! 移动到(7、8)
PRINT  distance                     ! 显示移动距离
END
!
MODULE  Move                        ! 将模块move置于主程序之后
    SHARE  x0,y0,x1,y1
    SUB  Start From(x,y)
         LET  x0,x1 = x
         LET  y0,y1 = y
    END  SUB
    SUB  MoveTo(x, y)
         LET  x0=x1
         LET  y0=y1
         LET  x1=x
         LET  y1=y
    END  SUB
    DEF  distance=sqr((x1-x0)^2+(y1-y0)^2)
END  MODULE
-------------------------
Ok.RUN
2.82843
7.07107
Ok.
```

在这个例子中，由于 x0，y0，x1，y1 是共享变量，因此它们是静态的，程序运行时它们的值一直保存在内存中。只要调用 StartFrom 或 MoveTo 修改了它们的值，修改后的值就能够在随后调用 StartFrom，MoveTo 或 distance 时使用。同时，因为 x0，y0，x1，y1 在模块中共享，不需要借助形式参数在过程之间传递它们的值，因此简化了形式参数表，使过程的调用简洁明了。

相反，如果不使用共享变量，x0，y0，x1，y1 就必须出现在每个过程的形式参数表中，主程序对其进行调用时必须相应给出 4 个与主程序本身无关的实际参数。

应该指出，使用模块中定义的公用变量时，在主程序和模块外的其他外部过程中，必须首先用 DECLARE PUBLIC 语句对这些公用变量加以声明。如果公用变量是一个数组，声明时则需要在数组名后带有圆括号和说明维数的逗号。例如，假设某个模块具有如下 PUBLIC 语句：

```
PUBLIC  a,b,x(10),y(10),t$(5,5)
```

则在主程序和模块外的其它外部过程中相应的 DECLARE PUBLIC 语句为：

```
DECLARE  PUBLIC  a,b,x( ),y( ),t$(,)
```

例 16 使用堆栈存取数据。

堆栈是程序设计中常用的一种数据结构。如果有一组数据，存入时的顺序是 x_1，x_2,……,x_{n-1}，x_n，而取出时的顺序是 x_n，x_{n-1}，……，x_2，x_1，即数据按先进后出的原则存取，那么这样的存取方式便是堆栈存取（图 9-6）。

为了实现堆栈存取。需要设计以下 3 个子程序：

init(n) 对堆栈初始化，并将堆栈尺寸(所能存储的数据个数)设置成 n；

push(x) 将变量 x 的值压入栈内，置于栈内所有数据的最上层，堆栈增加一层；

pop(x) 将堆栈最上层的数据弹出并赋给变量 x，弹出后堆栈降低一层。

另外，还需要设计一个与堆栈有关的变量 FreeStack，反映堆栈存储数据时剩余空间还能够存储数据的个数。

为了简化堆栈的使用和保证堆栈不受外界干扰，把有关堆栈的操作处理全部定义在模块 stack 内，然后将模块存放在库 stack.lib 中。

程序：

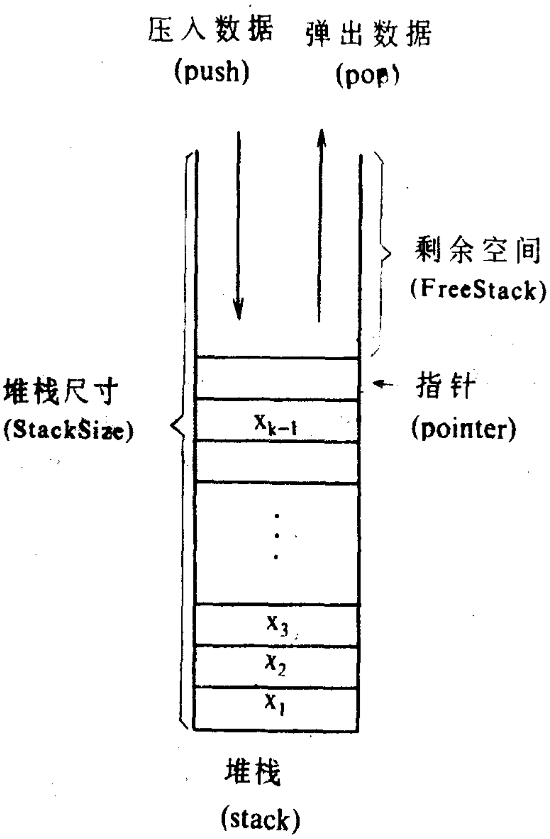

压入数据
(push)
弹出数据
(pop)
剩余空间
(FreeStack)
指针
(pointer)
堆栈尺寸
(StackSize)
x_{k-1}
x_3
x_2
x_1
堆栈
(stack)

图 9-6 堆栈数据存取示意

```
EXTERNAL
!
MODULE  Stack
PUBLIC  Free Stack
SHARE  Stack(100), Stack Size, pointer
PRIVATE  Stack Error
LET  pointer=0
LET  Stack Size, FreeStack=100
!
SUB  init(n)      ! 堆栈初始化
     LET  StackSize=n
     MAT  stack=zer(n)
END  SUB
!
SUB  push(x)   ! 数据入栈
     IF  pointer<StackSize  then
         LET  pointer=pointer+1
         LET  Free Stack=StackSize-pointer
         LET  Stack(pointer)=x
     ELSE
         CALL  StackError
     END  IF
END  SUB
!
SUB  pop(x)      ! 数据出栈
     IF  pointer>0 then
         LET  x=stack(pointer)
         LET  pointer=pointer-1
         LET  Free Stack= StackSize-pointer
     ELSE
         CALL  Stack Error
     END  IF
END  SUB
!
SUP  Stack Error     ! 出错时进行有关处理
     IF  pointer=Stack Size  then
         PRINT  "Stack Overflow !"
     ELSE
         PRINT  "Stack Empty !"
     END  IF
     STOP
END  SUB
END  MODULE
-----------------------------------
Ok.SAVE stack.lib
```

在模块 stack 中，将变量 FreeStack 定义成公用变量，以便从模块外能够得知堆栈的剩余空间。堆栈尺寸变量 StackSize，堆栈指针变量 pointer 和堆栈数组 stack 为模块内的共享变量，它们不受模块外部的影响。子程序 StackError 仅在用 push 或 pop 进行堆栈存取出错时，才用来显示出错信息及中断程序运行。显然 StackError 不应被模块外部调用，因此被定义成模块的私有过程。初始化后，堆栈尺寸的默认值设定为 100，除非调用 init 重新设置堆栈的尺寸。

以下程序利用库“stack.lib”中的模块 stack 进行堆栈存取：

```
LIBRARY  "stack.lib"                       ! 指定所用的库名
DECLARE  PUBLIC  FreeStack                 ! 声明FreeStack为公用变量
INPUT  prompt "How many data ?" :n         ! 输入堆栈的尺寸
CALL  init(n)                              ! 堆栈初始化
DO  while  FreeStack > 0                   ! 当堆栈未满时循环
    INPUT  x                               ! 输入一个数据
    CALL  push(x)                          ! 压入栈的顶部
LOOP
DO  while  FreeStack < n                   ! 当堆栈未空时循环
    CALL  pop(x)                           ! 弹出栈顶数据
    PRINT  x;                              ! 显示出该数据
LOOP
END
------------------------------------------
Ok.RUN
How many data ?  5
? 10
? 20
? 30
? 40
? 50
50  40  30  20  10
Ok.
```

通过此例可以看出，除了模块 stack 开放给外部使用的公用变量 FreeStack，子程序 init(n)，push(x)和 pop(x)以外，模块内部有关堆栈的操作处理细节对外部是隐蔽的。隐蔽细节而提供对数据结构进行访问的方式称为数据抽象，隐蔽细节而提供对运算功能进行调用的方式称为运算抽象。数据抽象和运算抽象是结构化程序设计中解决复杂程序设计问题的重要软件工程方法。

9.6 程序举例

例 17 二分法检索。

当数据量很大时，采用按顺序逐一查找的方法（顺序检索）是十分费时的。如果数据已预先排序，采用二分法检索技术则可实现数据的快速查找。

在已排序的一组数据中查找指定数据时，由于数据可能出现重复，因此一般要求查找的是第 1 个符合检索条件的数据。现假设一维数组 A 已按升序排列，要查找的数据为 x，并将查找范围的左、右边界的下标记为 l，r，则相应的算法如下：

令 $i=\mathrm{INT}((l+r)/2)$，计算中点的下标位置 i；

如果 $A(i)<x$，则第一个 x 的下标必然 $\geqslant i+1$，因此令 $l=i+1$ 使左下标边界右移，否则第一个 x 的下标必然 $\leqslant i$，因此，令 $r=i$ 使右下标边界左移；

重复上述过程直到 $l=r$，此时查找范围缩小至只剩下一个元素 $A(l)$；

如果 $A(l)=x$ 则 $A(l)$为第一个 x，否则数组 A 中不存在 x。

根据以上算法编制一个对一维数组进行二分法检索的子程序 find 。它的第 1 个形式参数对应一维数组，第 2 个形式参数对应要查找的数据，第 3 个形式参数用来送出查找到的数据的下标（如果未找到，则传出的值为数组下标的下界值减 1 ）。

程序：

```
SUB  find(a( ),x,d)
     LET  l=lbound(a)
     LET  r=ubound(a)
     DO
          LET  i=int((l+r)/2)
          IF  a(i)<x then  LET l=i+1  else  LET  r=i
     LOOP  until l=r
     IF  a(l)=x  then  LET  d=l  else  LET  d=lbound(a)-1
END  SUB
```

二分法还可用于字符串数据的快速检索。下面是根据学生的编号对学生的有关信息（如姓名，性别，年龄等）进行检索的例子。其中编号采用字符串的形式（例如 H86215 表示：化学系 86 级 2 班 15 号），程序首先将学生信息读入对应的数组，然后调用子程序 seek 对指定的编号进行二分法检索。检索时允许仅指定编号信息的左边某一部分（例如， H（化学系）、H86（化学系 86 级）、H862（化学系 86 级 2 班）等），从而灵活地改变检索的范围。

程序：

```
DIM  code$(8), nameS(8), sex$(8), age(8)
FOR  i=1 to 8
     READ  codeS(i), name$(i), sex$(i), age(i)
NEXT  i
DO
     INPUT  prompt "Code=? (0 to Exit) ":c$
     IF  cS ="0"  then  EXIT  DO
```

```
        LET  c$ =ucase$(c$)
        CALL  seek(code$,c$,index)
        DO  while index>0
            PRINT code S(index), name$(index), sex$(index), age(index)
            LET index=index+1
        LOOP  while  index<=8  and  code$(index)[1:len(c$)]=c$
        PRINT
LOOP
!编号须用大写字母，并根据编号的 ASCII 码顺序放置学生信息
DATA  H86215, li hua, m, 22
DATA  H86312, liu ming, m, 21
DATA  H87103, zhao qiang, m, 21
DATA  H87124, deng li, f, 19
DATA  S85120, he gang, m, 23
DATA  S86121, shun rong, f, 20
DATA  W86104, wang ling, f, 19
DATA  W86301, lin hong, m, 20
END
!二分法检索子程序
SUB  seek(a$( ), n$,d)
     LET  l=lbound(a$)
     LET  r=ubound(a$)
     DO
         LET  i=int((l+r)/2)
         IF  a$(i)[1:len(n$)]<nS  then  LET  l=i+1  else  LET  r=i
     LOOP  until  l=r
     IF  aS(l)[1:len(n$)]=nS  then  LET  d=l  else  LET  d=lbound(a$)-1
END  SUB
------------------------------------------------------------
Ok.RUN
Code=? (0 to Exit) h87
H87103            Zhang Qiang          m               21
H87124            Deng Li              f               19
Code=? (0 to Exit) s
S85120            He Gang              m               23
S86121            Shun Rong            f               20
Code=? (0 to Exit) 0
Ok.
```

例 18　利用平面网格化数据进行曲面插值。

第 8 章例 21 中介绍了具有一个自变量的一组数据的拉格朗日曲线插值法，现将其推广到有两个自变量的情形，即曲面插值。

假设平面区域[$x_{min}<x<x_{max}$; $y_{min}<y<y_{max}$]上有 r 条垂直于 x 轴的网格线和 c 条垂直于 y 轴的网格线。因此平面共有 $r\times c$ 个网格点。网格点的编号(i, j) 表示第 i 条垂直于 x 轴和第 j 条垂直于 y 轴的网格线交于该点($1<i<r$, $1<j<c$)。网格点的平面坐标记为(x_i, y_j)。

若所有网格点 (x_i, y_j) 上的函数值 $F(x_i, y_j)$ 均为已知，对于平面上的任意点(x, y), $F(x, y)$ 的值可按以下方法求出:

(1) 确定 x 的 3 个邻近的网格线 i_1, i_2, i_3 和相应的坐标值 x_1, x_2, x_3;

(2) 确定 y 的 3 个邻近的网格线 j_1, j_2, j_3 和相应的坐标值 y_1, y_2, y_3;

以上两步完成后，得出点(x, y)的 9 个邻近的网格点 $a_1(x_1, y_1)$, $a_2(x_2, y_1)$, $a_3(x_3, y_1)$, …, $a_8(x_2, y_3)$, $a_9(x_3, y_3)$

(3) 根据 3 个点的拉格朗日曲线插值公式，分别计算出 $f(b_1)$, $f(b_2)$和 $f(b_3)$, 然后再计算出 $f(x, y)$。其中:

利用 $f(a_1)$, $f(a_2)$, $f(a_3)$计算出 $f(b_1)$;

利用 $f(a_4)$, $f(a_5)$, $f(a_6)$计算出 $f(b_2)$;

利用 $f(a_7)$, $f(a_8)$, $f(a_9)$计算出 $f(b_3)$;

利用 $f(b_1)$, $f(b_2)$, $f(b_3)$计算出 $f(x, y)$。

由于每次计算所用的 3 点及所求的插值点总是位于同一直线上，因此可以化为一个自变量的情形来处理，每次仅考虑它们的横坐标或纵坐标。各点的位置关系如图 9-7 所示。

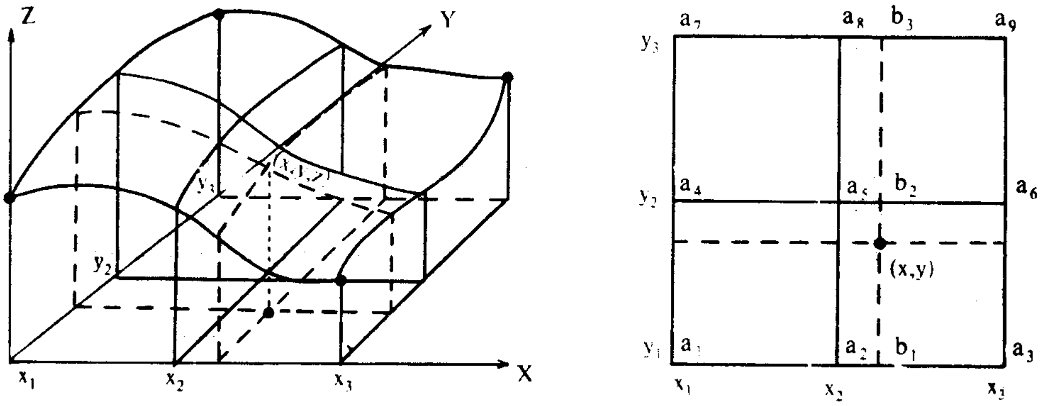


图 9-7 曲面插值示意与点位关系

下面按上述算法编制具体程序。程序中的函数 f 为曲面插值函数。调用函数 f 时，需代入存放平面网格化数据的数组、平面区域的范围 (x_{min}, x_{max}, y_{min}, y_{max})以及插值点的坐标 (x, y)。仅用 3 个点的拉格朗日曲线插值公式定义为函数 *lglr3*, 提供给曲面插值函数 f 调用。程序要求方向相同的网格线是等距的。

首先由函数 f 和函数 lglr3 建立库 insert.lib:

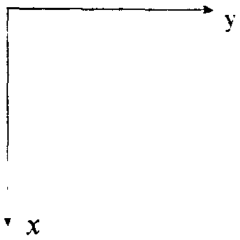
```
EXTERNAL
DEF  f(a(,),xmin,xmax,ymin,ymax,x,y)
     DECLARE DEF lglr3
     ! 确定 x 的 3 个邻近的网格线 i1, i2, i3 和相应的坐标值 x1、x2, x3
```

```
    LET  r=ubound(a,1)                  ! r为垂直于X轴的网格线的条数
    LET  dx=(xmax-xmin)/(r-1)           ! dx为垂直于X轴的网格线的间距
    LET  i=int((x-xmin)/dx)+1           ! i为x右侧(即xi<x)最近的网格线
    IF  i=r then
        LET  i1=r-2
    ELSE  IF  i=1 then
        LET  i1=1
    ELSE
        LET  i1=i-1
    END  IF
    LET  i2=i1+1
    LET  i3=i2+1
    LET  x1=xmin+(i1-1)*dx
    LET  x2=x1+dx
    LET  x3=x2+dx
    ! 确定y的3个邻近的网格线j1, j2, j3和相应的坐标值y1, y2, y3
    LET  c=ubound(a,2)                  ! c为垂直于y轴的网格线的条数
    LET  dy=(ymax-ymin)/(c-1)           ! dy为垂直于y轴的网格线的间距
    LET  j=int((y-ymin)/dy)+1           ! j为y下方(即yj<y)最近的网格线
    IF  j=c  then
        LET  j1=c-2
    ELSE  IF  j=1 then
        LET  j1=1
    ELSE
        LET  j1=j-1
    END IF
    LET  j2=j1+1
    LET  j3=j2+1
    LET  y1=ymin+(j1-1)*dy
    LET  y2=y1+dy
    LET  y3=y2+dy
    ! 调用lglr3计算插值点的值
    LET  fb1=lglr3(x1, x2, x3, a(i1,j1), a(i2,j1), a(i3,j1), x)
    LET  fb2=lglr3(x1, x2, x3, a(i1,j2), a(i2,j2), a(i3,j2), x)
    LET  fb3=lglr3(x1, x2, x3, a(i1,j3), a(i2,j3), a(i3,j3), x)
    LET  f=lglr3(y1, y2, y3, fb1, fb2, fb3, y)
END  DEF
!
DEF  lglr3(u1,u2,u3,v1,v2,v3,u)
    LET  f1=v1*(u-u2)*(u-u3)/((u1-u2)*(u1-u3))
    LET  f2=v2*(u-u1)*(u-u3)/((u2-u1)*(u2-u3))
    LET  f3=v3*(u-u1)*(u-u2)/((u3-u1)*(u3-u2))
```

```
        LET  lglr3=f1+f2+f3
    END  DEF
```

需要时，可直接调用库 insert.lib 中的函数 f:

```
    LIBRARY  "insert.lib"
    DECLARE  DEF  f
    DIM  a(1,1)
    READ  r,c,xmin,xmax,ymin,ymax
    MAT  READ  a(r,c)
    DO
        INPUT  prompt "x, y=?":x, y
        IF  x<xmin  or  x>xmax  or  y<ymin  or  y>ymax  then  EXIT  DO
        PRINT  f(a,xmin,xmax,ymin,ymax,x,y)
    LOOP
    !
    DATA  6, 6, 1, 6, 1, 6
    !
    DATA  2, 6, 5, 3, 7, 8          !
    DATA  3, 4, 4, 3, 5, 7          !
    DATA  5, 4, 3, 3, 4, 6          !
    DATA  5, 4, 3, 3, 3, 4          !
    DATA  4, 3, 3, 2, 2, 3          !
    DATA  5, 4, 2, 1, 0, 2          !
    END
    ----------------------------------------
    Ok.RUN
    x, y=? 4.5, 4.6
     2.52
    x, y=? 0, 0         (超出网格范围则退出)
    Ok.
```



程序中数据的行列下标分别对应于 x 轴和 y 轴的坐标。所以将网格点上的函数值置入 DATA 语句块中时，应把行作为 x 轴的方向（即 x 坐标不变），列作为 y 轴的方向（即 y 坐标不变）。

例 19　快速排序(图 9-8)。

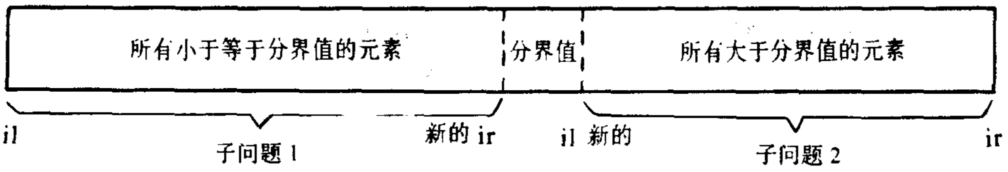


图 9-8　快速排序示意

设待排序数组 A 的下标上下界分别为 ir, il（显然有 $il<ir$）。进行快速排序时，首先在 A 中任选某一元素作为分界值（不失一般性，不妨将第一个元素作为分界值）。然后整理数

组，使所有小于等于分界值的元素调到分界值的左面，将所有大于分界值的元素调到分界值的右面。于是，将数组 *A* 整理成图 9-8 的形式。此时，整个数组的排序问题，便分解为对分界值左右两个部分数组的排序问题。于是，只要对这两个排序子问题，按上述方法递归处理，即可实现对整个数组的排序。

递归子程序：

```
SUB  FastSort(il,ir,a( ))
       IF  il<ir  then
            LET  t=a(il)                               ! 将第一个元素作为分界值 t
            LET  i=il                                  ! 用 l 标记其左侧的元素均已整理成<t
            LET  r=ir                                  ! 用 r 标记其右则的元素均已整理成>t
            DO                                         ! 循环地整理数组
                 DO  until  a(r)= <t or l=r            ! 从 r 向左寻找<t的元素
                      LET r=r-1                        ! 每找一次， r左移一位
                 LOOP
                 ! 如果未整理完毕则将<t 的元素调入 l 的位置
                 IF  l<r then
                     LET  a(l)=a(r)
                     LET  l=l+1                        ! 将 l 右移一位
                 END  IF
                 DO  until  a(l)>t or l=r              ! 从 l 向右寻找>t的元素
                      LET  l=l+1                       ! 每找一次，l 右移一位
                 LOOP
                 ! 如果未整理完毕则将>t 的元素调入 r 的位置
                 IF  l<r then
                      LET  a(r)=a(l)
                      LET  r=r-1                       ! 将 r 左移一位
                 END  IF
            LOOP  until  l=r                           !直至 l=r，循环整理过程结束
            LET  a(r)=t                                ! 此时，r 为分界值的位置
            CALL  FastSort(il,r-1,a)                   ! 递归处理分界值左边的部分
            CALL  FastSort(r+1,ir,a)                   ! 递归处理分界值右边的部分
    END  IF
END  SUB
```

若将这一子程序置入库 lib.tru 中，则对 n 个随机数进行快速排序的主程序可以编制为：

```
LIBRARY  "lib.tru"
DIM  a(1)
INPUT  prompt"n=":n
MAT  a=zer(n)
FOR  i=1 to n
       LET  a(i)=int(100*rnd)
NEXT  i
```

```
MAT  PRINT a,
PRINT  "start sortting.please wait……"
LET  begin=time
CALL  FastSort(1,n,a)
LET  finish=time
PRINT  "It took"  finish-begin;  "sec. for sortting."
PAUSE  3                  ! 暂停 3 秒钟
MAT  PRINT a,
END
-------------------------------------------
Ok.run
n=? 20
 54        7       68       33       95
 82       54        8       39       58
 75       51        3       42       49
 58       10       82       48       77
Start sortting.please wait…
It took 0.22 sec. for sorting.
  3        7        8       10       33
 39       42       48       49       51
 54       54       58       58       68
 75       77       82       82       95
Ok.
```

习 题 9

(1) 下面是将外币折换成人民币的计算程序．其中，自定义函数 v 返回每百单位外币对应的人民币值．试选择适当的内容填空．

```
DIM  s$(1), c(1)
!
DEF  v(name$)
     FOR  i=1 TO n
          IF  s$(i)=name$  then
              (           )
              EXIT  FOR
          END  IF
     NEXT  i
END  DEF
!
LET  n=5
(          )
```

```
MAT READ s$
MAT READ c
INPUT PROMPT "money, country=":money, name$
(           )
PRINT yuan
DATA "USA", "HK", "FRANCE", "ITALY", "JAPAN"
DATA 471.03, 60.33, 80.48, 0.3668, 3.2142
END
------------------------------------------------------------
```

A) CALL yuan＝c(n)＊money／100
B) LET yuan＝v(name$)＊money／100
C) LET yuan＝v(name$)＊money＊100
D) LET yuan＝v(name$)
E) CALL v(name$)＊money／100
F) CALL yuan＝c(n)
G) LET c＝v
H) LET v＝c(n)
I) LET v＝c(i)
J) LET v(name$)＝c(n)
K) LET v(name$)＝c(i)
L) DIM s$(n), c(n)
M) MAT REDIM s$(n), c(n)
N) MAT PRINT s$; c;

(2) 试编写函数 COUNTSAME，其自变量 x，y，z 均为字符型。若所有字符均不相同，则函数取零值；仅两个相同，则取值 1；若全相同，则取值 2。

(3) 下面的英文式子

$$\begin{array}{r} \text{PEACH} \\ +\ \text{LEMON} \\ \hline \text{APPLE} \end{array}$$

译成中文是：桃子+柠檬＝苹果，请用数字 1～9 去替换字母，使算式仍然成立（相同字母表示相同数字，不同字母表示不同数字），编程序找出这些字母所对应的数字。本题有多解。提示：注意考虑进位问题。

(4) 编程计算 $Y=f(u+v)/(f(u)+f(v))$，其中，$f(t)=(a\sin(t)+b\cos^2(t))/4$，$a$ 和 b 为常数，u 和 v 为变量。

(5) 使用函数计算下列各积分：

① $\int_{1}^{2.5}\frac{\sin(x)}{\sqrt{1-x^2}}dx$　　② $\int_{2}^{3.7}\frac{e^{x}-e^{-x}}{2}dx$

(6) 函数 A 定义如下：$A(0,\ n)=n+1$；$A(m,\ 0)=A(m-1,\ 1)$；$A(m,\ n)=A(m-1,\ A(m,\ n-1))$；编程计算 A。

第10章　图　形

信息的形式是多种多样的。除了数字和文字符号外，图形也是一种重要的信息形式。图形显示已经是普通计算机的基本输出方式之一。它是将计算机的运算处理结果，以图形的直观形式，输出到计算机的显示屏幕或其他外部设备上。

True BASIC 具有较强的屏幕图形显示功能。利用 True BASIC 的绘图语句，可以在屏幕上按任意设定的直角坐标系来画点、线、面和着色，并由此组成各种复杂图形。True BASIC 不仅允许利用绘图语句构成各种绘图子程序，而且还允许定义一种专用于绘图的逻辑模块——图画。调用图画时能够对图形进行平移、旋转、放大等变换。此外，本章还介绍 True BASIC 的动画、多窗口显示、菜单等功能，这些都是现代应用程序设计中被广泛运用的新技术。

10.1　屏幕模式与窗口设置

在 True BASIC 基本屏幕状态下，显示器分为上下两个窗口：上方的编辑窗口显示程序，下方的命令窗口显示操作命令和运行结果。除基本屏幕状态以外，True BASIC 还具有几种可以在程序中设置的屏幕模式。当程序运行中使用这些模式时，编辑窗口和命令窗口暂时消失，整个屏幕作为完整的窗口显示数据或图形。程序结束或运行出错后，屏幕会自动退出所用模式并返回到基本状态。

10.1.1　屏幕模式

大部分计算机所显示的图形，都是由一行行整齐排列的像素构成的光栅图形。光栅图形的一个像素对应于屏幕的一个光点。屏幕上光点的密度即为图形的分辨度。图形的显示质量与分辨度有关：分辨度越高，显示的图形便越精细。

图形的分辨度取决于屏幕的模式。True BASIC 支持的屏幕模式主要有：

(1) 正文模式(或称为文本模式) 用于显示正文(即程序的数值计算或字符处理的结果)。屏幕可显示 25(行)× 80(列)个字符，可以同时使用 16 种颜色。

(2) 中分辨度图形模式(简称图形模式) 用于显示图形和正文。图形的分辨度为 320(水平方向像素个数)× 200 (垂直方向像素个数)，正文可显示的行列数为 25(行)× 40(列)。可同时使用 4 种颜色。

(3) 高分辨度图形模式(简称高分辨度模式) 用于显示精度较高的图形和正文。图形的分辨度为 640×200，正文可显示的行列数为 25×80。只能同时使用 2 种颜色。

以上三种屏幕模式适用于配置 CGA 彩色显示器的计算机。如果计算机配置的是 EGA 彩色显示器，除此之外还可使用以下屏幕模式：

(4) EGA 高分辨度图形模式(简称 EGA 高分辨度模式) 用于显示高精度图形和正文。图形的分辨度为 640×350，正文可显示的行列数为 25×80。可同时使用 16 种颜色。

如果计算机配置 HERCULES 单色显示器，绘图时则必须使用以下屏幕模式：

(5) HERCULES 模式用于显示高精度单色图形和正文。图形的分辨度为 720×348，正文可显示的行列数为 25×80。

实际上，由于计算机硬件性能的增强，某些计算机具有更高的屏幕分辨度和更多的颜色

种类。

屏幕模式设置语句格式：

SET MODE 模式名

作用：将屏幕设置成指定的模式。其中模式名为下列字符串之一：

“80”	(正文模式)
“Graphics”	(图形模式)
“Hires”	(高分辨度模式)
“Egahics”	(EGA 高分辨度模式)
“Hercules”	(HERCULES 模式)

模式名的字符串中对字母大小写不作区分。

在程序运行过程中，屏幕模式一经设置就保持其状态，直到被重新设置成另一模式为止。如果在执行正文模式的各种语句之前没有设置正文模式，True BASIC 将会自动设置“80”模式。如果在执行各种绘图语句之前，没有设置与图形相关的屏幕模式，True BASIC 则会自动将屏幕设置成“Graphics”模式(对于 HERCULES 单色显示器则自动设置成“Hercules”模式)。

如前所述，在一台计算机上能够设置哪些屏幕模式，取决于显示器的类型。本教程的图形部分主要是针对配置 CGA 彩色显示器的计算机编写的，因此其中的程序示例仅使用“80”，“Graphics”，“Hires”三种屏幕模式。对于配置其他显示器的计算机，只需在 SET MODE 语句中改设成相应的屏幕模式即可。

10.1.2 窗口设置

在 True BASIC 中，图形是用直角坐标系来表示的。由于屏幕尺寸有限，因此在与图形有关的各种模式下，屏幕被看成是直角坐标系中的一个矩形部分。这个矩形部分称为窗口。显示图形时，只显示位于窗口之内的局部图形，超出屏幕范围的图形将被忽略。

为了使整个图形位于窗口之内，需要合理地设置窗口。设置窗口就是确定窗口在直角坐标系中的位置和大小，即说明屏幕表示的是直角坐标系中的哪一个矩形部分。

窗口设置语句格式：

SET WINDOW x_{min}, x_{max}, y_{min}, y_{max}

作用：将屏幕窗口设置成直角坐标系下的矩形部分($x_{min}<x<x_{max}$; $y_{min}<y<y_{max}$)。

其中，x_{min}, x_{max}, y_{min}, y_{max} 均为数值表达式，分别表示窗口内 x，y 坐标的最小值和最大值(图 10-1)。

窗口设置后，屏幕 4 个顶点的坐标为：(x_{min}, y_{min})；(x_{max}, y_{min})；(x_{max}, y_{max})；(x_{min}, y_{max})。

若在执行各种绘图语句之前没有设置窗口，True BASIC 将按默认值自动设置窗口。默认的窗口设置为：SET WINDOW 0，1，0，1。

具体显示器的屏幕尺寸是固定的，因此当窗口设置不同时，屏幕上 x 轴和 y 轴的几何比例也不一样。标准的 IBM-PC 显示器的屏幕是长方形的，其高度与宽度之比约为 2 : 3.2. 这个比例称为屏幕的纵横比。不同显示器的屏幕纵横比往往不同。因此对于具体计算机的显示器，应当根据其屏幕纵横比来调整设置窗口时所用的参数，以便正确反映图形的形状。

假设屏幕纵横比为 2／3，图 10-2 是在不同的窗口参数下显示半径为 10 的圆时所得到的

图形:

当窗口设置为 SET WINDOW −15，15，−15，15 时，窗口的高宽比为 1，大于屏幕的纵横比。此时 y 轴上的单位长度被压缩，因而图形为一扁椭圆(图 10−2(a))。

当窗口设置为 SET WINDOW −15，15，−10，10 时，窗口的高宽比为 2／3，等于屏幕的纵横比，因而图形为一正圆(图 10−2(b))。

当窗口设置为 SET WINDOW −20，20，−10，10 时，窗口的高宽比为 1／2，小于屏幕的纵横比。此时 X 轴上的单位长度被压缩，因而图形为一长椭圆(图画 10−2(c))。

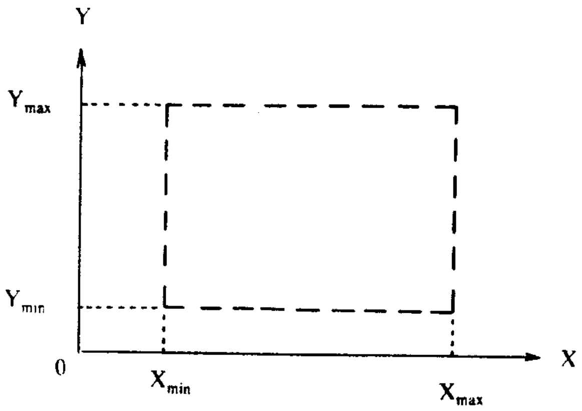
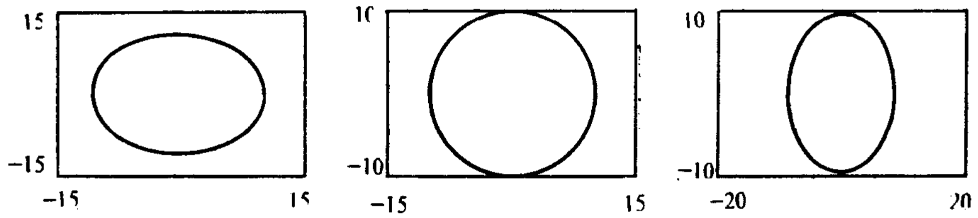


图 10−1　屏幕窗口的范围



(a)　(b)　(c)

图 10−2　窗口参数对图形显示的影响

10.2　基本绘图语句

任何复杂的图形都由点、线、面这些基本的图形元素组成。True BASIC 的基本绘图语句包括：画点语句、画线语句、画面语句及写字符语句。这些基本绘图语句以及后面还将介绍的其他绘图语句，都应当在与图形有关的模式下执行。

10.2.1　画点语句

语句格式:

PLOT POINTS: x_1，y_1；x_2，y_2；…；x_n，y_n

作用：按给定的坐标(x_i，y_i)在屏幕上画点. 当给出两个以上的点的坐标时，各点的坐标对之间以分号“;”相隔。

例 1　在屏幕上用点绘出一个圆。

圆的参数方程为

$$x = a \times \cos t$$
$$y = a \times \sin t$$

设 $a=1$，t 由 0 变化到 2π。

程序:

```
SET MODE ″Graphics″
SET  WINDOW  -1.5, 1.5, -1, 1
FOR  t=0  to 2*pi  step  pi/50
     PLOT  POINTS:cos(t), sin(t)
NEXT  t
END
```

如果在程序中设置了屏幕模式，那么程序运行结束后，屏幕上仍将保持程序运行时产生的图形或正文. 直至按下任意一键，屏幕才恢复到基本状态。

需要注意的是：由于分辨度的关系，如果点的坐标值十分接近，那么将把这些点显示在屏幕的同一光点位置上。

10.2.2 画线语句

语句格式：

PLOT　LINES: x_1, y_1; x_2, y_2; …; x_n, y_n

作用：将给定坐标值的一组点 (x_i, y_i)，按顺序用直线段连成一条折线. 相邻点的坐标对之间以“；”号相隔.

例如，语句

```
PLOT  LINES : 0, 0 ;  0, 1 ;  1, 1 ;  1, 0 ;  0, 0
```

将绘出一个矩形. 如果窗口参数选择恰当，则这个矩形为单位正方形. 而语句

```
PLOT  LINES : 0, 0 ;  1, 1
```

则显示由两点连成的一段直线.

一个 PLOT　LINES 语句也可以拆成若干句来书写。比如，上面画长方形的语句可以分写成四个语句，其中前 3 个语句需后接分号“;”:

```
PLOT  LINES : 0, 0 ;
PLOT  LINES : 0, 1 ;  1, 1;
PLOT  LINES : 1, 0 ;
PLOT  LINES : 0, 0
```

语句后接分号，表示折线还将延续. 不接分号，则表示折线终止于本语句给出的最后一个点上. 因此，通常采用连续执行一组后接分号的 PLOT　LINES 语句，或者循环执行一个后接分号的 PLOT　LINES 语句的方法，来绘制一条连续的折线。如果在执行了一个后接分号的 PLOT　LINES 语句之后需要终止折线，可使用一个不带点坐标的 PLOT　LINES 语句.

例 2　在屏幕上用折线绘制一组圆. 圆的半径分别取 0.4, 0.6, 0.8, 1.

程序：

```
SET  MODE  "Graphics"
SET  WINDOW  -1.5, 1.5, -1, 1
FOR  a=0.4  to  1  step  0.2                          ! 循环给出不同的圆半径
     FOR  t=0  to  2*pi  step  pi/50                  ! 循环画出圆的各段直线
          PLOT  LINES: a*cos(t), a*sin(t);            ! 每段折线都向后延续
     NEXT  t                                          ! 画下一段直线
     PLOT  LINES                                      ! 画完一个圆后终止折线
NEXT  a                                               ! 画下一个圆
END
```

PLOT　LINES 语句可以简写成:

$$\text{PLOT}\quad x_1,\ y_1;\ \ x_2,\ y_2;\ \ \cdots;\ \ x_n,\ y_n$$

10.2.3　画面语句

True BASIC 的画面语句需要给出面的边界点。面的边界为由边界点顺序连接而成的闭合折线。画出的面即为闭合折线围成的内部区域。

语句格式:

$$\text{PLOT}\quad \text{AREA}: x_1,\ y_1;\ \ x_2,\ y_2;\ \ \ldots;\ \ x_n,\ y_n$$

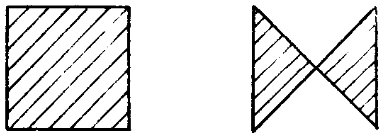
作用: 用直线段将各点依次连成闭合折线，然后用当前颜色充填闭合折线的内部区域。

在 PLOT AREA 语句中至少应当给出三个边界点，并且按照在边界上的前后顺序(顺时针或逆时针都可)写出这些点的坐标。以下两个 PLOT AREA 语句具有相同的边界点，但由于点的顺序不同，画出的面也不一样:

PLOT　AREA: 0, 0; 0, 1; 1, 1; 1, 0 和 PLOT AREA: 0, 0; 1, 1; 1, 0; 0, 1

这两个语句画出的图形分别为图 10-3 的(a)和(b)。

执行 PLOT AREA 语句时，如果由各点连成的闭合折线出现复杂的交叉情况，True BASIC 将按照以下规则确定屏幕上哪些点属于闭合折线的内部区域: 从某点出发向屏幕外引一条直线，若直线与闭合折线相交奇数次，则该点属于闭合折线的内部区域; 若相交偶数次则该点属于外部区域 (图 10-4)。



(a)　　(b)

图 10-3　边界点顺序对图形的影响

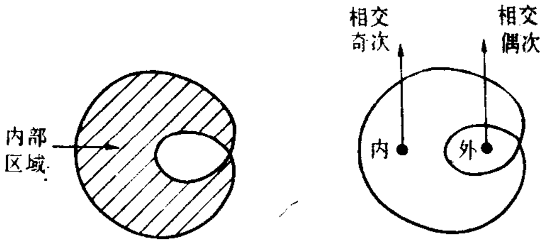


图 10-4　域内点与域外点的确定

10.2.4　用数组数据画点、线与面

如果在一个数组中事先存有一组点的坐标值，那么可以利用这个数组直接画出这组点，或者由这组点构成的折线、面。

用数组数据画点、线、面的语句格式分别为:

MAT　PLOT　POINTS : array

MAT　PLOT　LINES : array

MAT　PLOT　AREA : array

其中，直接用于画点、线或面的数组 array 必须是 $N\times 2$ 的数值数组，且第 i 行第 1 列存放第 i 点的 x 坐标，第 i 行第 2 列存放第 i 点的 y 坐标。

例 3　用 MAT　PLOT　LINES 语句画李萨茹曲线图。李萨茹曲线方程为

$$x=\sin(a\,t)$$

$$y=\cos(b\,t)$$

其中: $0<t<2\pi$　设 $a=3$, $b=5$。

程序:

```
！定义一个能存放 201 个点的数组 pointset
DIM  pointset(0:200, 2)
！将曲线上 201 个点的坐标存入 pointset
LET  d=pi/100
FOR  i=0 to 200
     LET  t=i*d
     LET  pointset(i, 1)=sin(3*t)
     LET  pointset(i, 2)=cos(5*t)
NEXT  i
！利用 pointset 画折线
SET  MODE  "Hires"
SET  WINDOW  -1, 1, -1, 1
MAT  PLOT  LINES:pointset
END
```

在此例中，由于折线多次相交，如果使用 MAT PLOT AREA 语句，则会显示出多个交错分散的面。

10.2.5 在直角坐标系下定位显示字符

PRINT 语句在与图形有关的各种屏幕模式下仍然有效，同样可用于显示正文。用它显示正文时，仍然按从左到右、从上到下的行列顺序进行显示，不能根据直角坐标系来指定显示的位置。因此，如果需要将有关的数字或字符与图形结合在一起显示，使用 PRINT 语句是不方便的。此时，可以用在直角坐标系下定位显示字符串的语句来实现。

语句格式：

```
PLOT  TEXT, AT  x, y : expr$
```

作用：定位显示字符串 expr$。显示时，expr$ 中第一个字符的左下角定位于坐标 (x, y) 处。

例 4 在直角坐标系下绘制正弦曲线的图形。要求标注直角坐标系的原点、x 轴、y 轴以及曲线的函数名称。

程序：

```
SET  MODE  "Graphics"                        ！设置模式
SET  WINDOW  0, 2*pi, -1, 1                  ！设置窗口[0, 2π; -1, 1]
PLOT  LINES:0, 0; 2*pi, 0                    ！画 x 轴
PLOT  LINES:0, -1; 0, 1                      ！画 y 轴
PLOT  TEXT, AT  0.1, 0.1:"0"                 ！标注原点
PLOT  TEXT, AT  2*pi-0.1, 0.1:"x"            ！标注 x 轴
PLOT  TEXT, AT  0.1, 0.9:"y"                 ！标注 y 轴
FOR  x=0 to 2*pi  step  pi/18                ！绘[0, 2π]上的正弦曲线
     PLOT  LINES:x, sin(x);
NEXT  x
PLOT  TEXT, AT  pi, 0.5:"y=sin(x)"           ！标注曲线
END
```

实际上，使用 PLOT TEXT 语句时仍然是将字符串显示在标准的行列位置上。对于图

形模式和高分辨度模式，屏幕的字符行列数分别为 25×40 和 25×80。定位显示时，语句 PLOT TEXT, AT x, y : expr$ 是将 expr$ 显示在 y 以下的最近一行和 x 以左的最近一列的位置上。

PLOT TEXT 语句将位置坐标(x, y)默认为是 expr$ 的左下角。如果需要在定位显示时按 expr$ 的其他部位来确定显示位置，则可用语句

SET TEXT JUSTIFY horiz$, vert$

加以说明。其中，horiz$，vert$ 分别表示定位显示时，x 和 y 以 expr$ 的哪一部位为标准。

horiz$ 可以是“Left”(左侧)、“Center”(中部)、“Right”(右侧)之一；vert$ 可以是“Bottom”(底部)、“Half”(中部)、“Top”(顶部)、“Base”(基部，即大写字母的底线)之一。例如，以下语句使得在显示字符串时利用字符串的中心来定位：

SET TEXT JUSTIFY “Center”，“Half”

而以下语句使得在显示字符串时利用字符串的右上角来定位：

SET TEXT JUSTIFY “Right”、“Top”

10.2.6 按行列值定位显示正文

定位显示除了可使用 PLOT TEXT 语句作外，还可按行列值定位光标后再使用 PRINT 等语句显示数字或字符。

光标定位语句格式：

SET CURSOR 行位置，列位置

作用：将光标定位到屏幕指定的行列位置上，使数字或字符能够在指定的行列位置上显示。该语句等价于 PRINT tab(行位置，列位置)语句。

例 5 模拟物体的抛射，显示物体抛射时的运动轨迹。

设 v_0(米／秒)为物体的初速度，α(度)为抛射角，t(秒)为时间，则物体运动轨迹的方程为：

$$x(t) = v_0 t\cos(\alpha)$$

$$y(t) = v_0 t\sin(\alpha) - 1/2gt^2$$

其中，$g = 9.8$(米／秒^2)为重力加速度. 根据 $y(t) = 0$ 解出物体落地的时刻为：

$$t = 2 \times v_0 \times \sin(\alpha) / g$$

程序：

```
OPTION ANGLE degrees
SET MODE "hires"
SET WINDOW 0, 100, -10, 50
! 画抛射点位置的垂线和地平线
PLOT LINES:0, 0; 0, 50
PLOT LINES:0, 0; 100, 0
! 定义抛射轨迹函数
DEF x(t)=v0 * t * cos(a)
DEF y(t)=v0 * t * sin(a)-9.8 * t * t / 2
! 循环抛射，i 为抛射次数
```

```
LET  i=0
DO
    LET  i=i+1
    ! 在屏幕右上角定位输入 v0, a
    SET  CURSOR  i, 60
    INPUT  prompt  "v0, a=?":v0, a
    IF  a<=0 or a>=90  then  EXIT  DO
    ! 将抛射轨迹分成 100 段画出
    LET  t1=2*v0*sin(a)/9.8
    PLOT  LINES:0, 0
    FOR  p=1  to  100
          LET  t=p*t1/100
          PLOT  LINES:x(t), y(t);
    NEXT  p
    PLOT  LINES
    ! 在落地点下方标出射程
    PLOT  TEXT, AT  x(t1), -5:str$(round(x(t1), 2))
LOOP
END
----------------------------------------------
Ok.RUN
v0, a=?30, 45
v0, a=?32, 70
v0, a=?0, 0
Ok.
```

程序运行时，在屏幕的右上方询问 v_0 和 α 的值。从键盘输入 v_0 和 α 的数值后，屏幕上显示相应的运动轨迹和落点距离。本程序可循环多次模拟抛射。当 v_0 和 α 的输入值均为 0 时，则程序运行结束。

10.3 图形着色

如果计算机配置彩色显示器，那末屏幕上的图形可以是彩色的。彩色图形不仅生动鲜明，而且图形包含的信息更直观、更易于理解。

10.3.1 不同屏幕模式下的颜色规定

显示彩色图形时，整个屏幕的色彩分为前景色和背景色。

前景色——即屏幕上图形或字符的颜色；背景色——即屏幕的底色。

在不同的屏幕模式下，可当作前景色或背景色的颜色种类是不同的。下面说明不同模式对前背景颜色的规定。

(1) 正文模式　在此模式下，可以为每一个要输出的字符串或数值设置各自的前景色和背景色。前景色可在表 10-1 的 16 种颜色中任选，背景色可在表 10-1 的前 8 种颜色（编号

0~7）中任选。

表 10-1　正文模式颜色

编号	名　称	含　义	编号	名　称	含　义
0	"BLACK"	黑	8		灰
1	"BLUE"	兰	9		亮　兰
2	"GREEN"	绿	10		亮　绿
3	"CYAN"	品兰	11		亮　品
4	"RED"	红	12		亮　红
5	"MAGENTA"	品红	13		亮品红
6	"YELLOW"	黄	14		亮　黄
7	"WHITE"	白	15		亮　白

(2) *图形模式*　在此模式下，可以为每一个图形光点或字符数据设置各自的前景色。整个屏幕取同一背景色。前景色分为两组，每组都有 4 种颜色。绘图时可以使用两组前景色中的一组(表 10-2)。

表 10-2　图形模式颜色

编号	名　称	编号	名　称
0	"BACKGROUND"	0	"BACKGROUND"
1	"GREEN"	4	"CYAN"
2	"RED"	5	"MAGENTA"
3	"YELLOW"	6	"WHITE"

每组前景色中的第一种颜色"BACKGROUND"即为当前的背景色，它的实际颜色取决于当前屏幕的背景色是何种颜色。将前景色设置成当前背景色时，由于所绘图形与背景同色，因此可以起到清除屏幕上原有图形或字符的作用。如果没有设置前景色，则系统自动认为前景色为"GREEN"。

当使用两组颜色中的一组绘图时，如果同时用到另一组中的颜色，那么图形原来的颜色将自动变换成另一组中的对应颜色(表 10-2)。

图形模式下可使用的背景色与正文模式中的背景色相同，即可在表 10-1 的前 8 种颜色中任选。

(3) *高分辨度模式*　在此模式下，屏幕上只允许使用两种颜色。背景色固定为黑色，前景色只能从表 10-1 列出的 16 种颜色中选用一种。

(4) EGA 高分辨度模式　这种模式可同时将表 10-1 的 16 种颜色用作前景色。整个屏幕的背景色则可以在前 8 种颜色中任选。EGA 高分辨度模式除了颜色表中规定的颜色外，还允许通过分别设置红、兰、绿三个基色的强度来合成所需要的颜色。

10.3.2　颜色的设置

设置前景色或背景色时，应当使用前景色或背景色的设置语句。无论在哪一种屏幕模式下，前景色和背景色设置语句的格式都不变。

设置前景色使用 SET　COLOR 语句。

语句格式:

SET　COLOR 名称或编号

作用：将屏幕的前景色设置成由所给出的名称或编号所指定的颜色。

例如，在图形模式下，SET　COLOR　“RED”和 SET　COLOR 2 都是将当前的前景色设置成红色。

只要设置了某种前景色，其后的绘图语句均以指定的颜色作图。若在绘图过程中不断改变前景色，则可得到由多种颜色构成的彩色图形。

设置背景色使用 SET　BACK 语句。

语句格式:

SET　BACK　名称或编号

作用：将屏幕的背景色设置成由所给出的名称或编号所指定的颜色。

例如，在正文或图形模式下，SET　BACK　“black”和 SET　BACK 0 都是将当前背景色设置成黑色。

例 6　绘制直角坐标系下阻尼振动曲线的彩色图形。绘图时使用图形模式的第一组颜色。其中，坐标系为红色，函数曲线为绿色，函数名称为黄色，背景为白色。

程序:

```
SET  MODE  "graphics"
SET  WINDOW  0, 2 * pi, -1, 1
SET  BACK  "white"
SET  COLOR  "red"
PLOT  LINES:0, 0; 2 * pi, 0
PLOT  LINES:0, -1; 0, 1
PLOT  TEXT, AT  0.1, -0.1:"0"
PLOT  TEXT, AT  2 * pi-0.1, -0.1:"x"
PLOT  TEXT, AT  0.1, 0.9:"y"
SET  COLOR  "green"
FOR  x = 0 to 2 * pi  step 0.05
     PLOT  LINES:x, exp(-x / 2) * sin(pi * x);
NEXT  x
SET  COLOR  "yellow"
PLOT  TEXT, AT  pi / 2, 0.6:"y = exp(-x / 2) * sin(pi * x)"
END
```

例 7　绘制单自变量数据的变化曲线。

由于这是常用的绘图形式，因此可以设计一个专用子程序 curve，以便需要时调用。为简单起见，令自变量 x 仅取整数值。因此绘图数据可以用一维数组 y(x)表示，其中自变量 x 当作下标使用。

子程序 curve 的算法:

(1) 求出 y 的最大值 y_{max}(设 > 0)；(2) 根据 y_{max} 设置窗口；(3) 画出坐标网格；(4) 标注自变量的取值范围；(5) 画出曲线。

子程序 curve:

```
SUB  curve(y( ))
    ！确定自变量的范围
    LET  xmin＝lbound(y( ))
    LET  xmax＝ubound(y( ))
    ！求出 y 的最大值 ymax
    LET  ymax＝y(xmin)
    FOR  x＝xmin to xmax
         LET ymax＝max(ymax， y(x))
    NEXT  x
    ！根据 ymax 设置窗口
    SET  WINDOW  xmin， xmax， −0.1 * ymax， 1.2 * ymax
    ！画出坐标网格
    SET  COLOR  ″red″
    FOR  x＝xmin to xmax                        ！画网格竖线
         PLOT  LINES:x， 0； x， 1.2 * ymax
    NEXT  x
    FOR  v＝0 to 1.2 * ymax  step  10           ！画网格横线
         PLOT  LINES:xmin， v； xmax， v
    NEXT  v
    PLOT  LINES:xmin， 1.2 * ymax； xmax， 1.2 * ymax
    ！标注自变量的取值范围
    SET  COLOR  ″yellow″
    PLOT  TEXT， AT xmin， −0.1 * ymax:str$(xmin)&″−″&str$(xmax)
    ！画出曲线
    SET  COLOR  ″green″
    FOR  x＝xmin to xmax
         PLOT  LINES:x， y(x)；
    NEXT  x
END  SUB
```

假设绘图数据为 1960 至 1979 之间各年度的降雨次数. 以年度为自变量绘制年降雨次数的变化曲线。

主程序:

```
DIM  value(0)
READ  FirstYear， LastYear
MAT  READ  value(FirstYear: LastYear)
SET  MODE  ″Graphics″
CALL  curve(value( ))
!
DATA  1960， 1979
DATA  14， 66， 17， 22， 35， 43， 46， 70， 52， 90
DATA  59， 15， 64， 6， 73， 72， 88， 50， 96， 67
END
```

10.3.3 填充颜色

在绘图过程中需要将图形的某个区域填上颜色时，可以首先画出这个区域的边界，然后用填色语句在区域内部填色。

填色语句格式:

```
FLOOD x, y
```

作用：从点(x，y)开始向四周扩散前景色，直至用前景色填满包含点(x，y)的区域。

使用 FLOOD 语句填色时，区域的边界应当闭合，否则填色过程将穿过边界的缺口，填到区域的外部。

例 8 绘制扇形图。

在统计分析中，常用扇形图来表示各项数据在数据总和中所占的比重。扇形图是由多个扇形组成的一个圆，因此首先定义画扇形的子程序 sector。 sector 形式参数的含义为:

a0 —起始角； a1 —终止角； c0 —边界颜色号； c1 —内部颜色号

填充内部颜色时，从扇形内部的点($0.5 * \cos((a0+a1)/2)$ ， $0.5 * \sin((a0+a1)/2)$)开始向四周扩散。另外，在扇形外侧标注百分比时，标注位置取为($1.2 * \cos((a0+a1)/2)$， $1.2 * \sin((a0+a1)/2)$)。

子程序 sector:

```
SUB  sector(a0, a1, c0, c1)
     SET  COLOR  c0
     PLOT  LINES:0, 0;
     FOR  a=a0  to  a1  step(a1-a0)/36
          LINES:cos(a), sin(a);
     NEXT  a
     PLOT  LINES:0, 0
     SET  COLOR  c1
     LET  m=(a0+a1)/2
     FLOOD  0.5*cos(m), 0.5*sin(m)
     LET  percent=round(100*(a1-a0)/(2*pi))
     SET  TEXT  JUSTIFY  "Center", "Half"
     PLOT  TEXT, AT  1.2*cos(m), 1.2*sin(m):str$(percent)&"%"
END  SUB
```

主程序:

```
DIM  v(0)
READ  n
MAT  READ  v(n)
LET  s=0
FOR  i=1 to n
     LET  s=s+v(i)
NEXT  i
!
SET  MODE  "Graphics"
SET  WINDOW  -2, 2, -1.3, 1.3
LET  t=0
FOR  i=1  to  n
     CALL  sector(2*pi*t/s, 2*pi*(t+v(i))/s, 3, mod(i, 2)+1)
     LET  t=t+v(i)
NEXT  i
DATA  8
DATA  245, 176, 83, 132, 50, 201, 34, 128
END
```

10.4　BOX 语句与动画设计

True BASIC 提供了一系列以 BOX 为前缀的语句。这些语句所完成的操作，都限制在屏幕上一个可任意指定其大小的矩形“盒子”(BOX)内。利用相应的 BOX 语句，可在屏幕上直接画出各种矩形和椭圆，因而简化了绘制矩形和椭圆的编程过程。BOX 语句还可以保存屏幕上某个矩形区域内的图形，并能够将被保存的图形重新显示出来。因此，BOX 语句可用于动画设计。

10.4.1　BOX 语句

设 x_{min}，x_{max}，y_{min}，y_{max} 为矩形在 x 轴和 y 轴上的最小值和最大值。BOX 语句将通过设置这些坐标值来确定矩形的大小和位置。

(1) 画空心矩形。语句格式:

BOX　LINES　x_{min}，x_{max}，y_{min}，y_{max}

作用：画给定矩形的边框。

(2) 画实心矩形。语句格式:

BOX　AREA　x_{min}，x_{max}，y_{min}，y_{max}

作用：画给定矩形并填充其内部。

(3) 画矩形的内切椭圆。语句格式:

BOX　ELLIPSE　x_{min}，x_{max}，y_{min}，y_{max}

或

BOX　CIRCLE　x_{min}，x_{max}，y_{min}，y_{max}

作用：画以矩形的四边为切线的内切椭圆。当矩形为正方形时；所画椭圆为一正圆。

(4) 清除矩形范围内的图形。语句格式:

BOX　CLEAR　x_{min}，x_{max}，y_{min}，y_{max}

作用：清除矩形范围内的图形。即用背景色画实心矩形。

(5) 将矩形范围内的图形存入字符串。语句格式:

BOX　KEEP　x_{min}，x_{max}，y_{min}，y_{max}　IN var$

作用：将矩形范围内的图形存入字符串变量 var$ 中。

(6) 将字符串中的图形在指定位置上显示。语句格式:

BOX　SHOW　var$　AT　x_{min}，y_{min}

作用：重新显示保存在 var$ 中的图形。显示时图形的左下角位于(x_{min}，y_{min})。

(7) 清屏。语句格式：

CLEAR

作用：清除整个屏幕，使全屏幕均为当前背景色。

例 9 绘制一个带有数字标记的直方图。已知数据为

30.1，41.8，96.3，65.2，23.9，70，34.2

程序(其中 bar 为绘制直方图的子程序)：

```
DIM  v(0)
READ  n
MAT  READ  v(n)
SET  MODE  ″graphics″
CALL  bar(v( ))
DATA  7                                          ! 数据的个数
DATA  30.1，41.8，96.3，65.2，23.9，70，34.2
END
!
SUB  bar(value( ))
     LET  n=size(value( ))
     SET  WINDOW  0，n+1，-10，150               ! 根据n设置窗口(设0<数据<100)
     SET  COLOR  ″yellow″
     BOX  LINES  0，n+1，0，150                  ! 画图框
     SET  TEXT  JUSTIFY  ″center″，″bottom″
     FOR  i=1 to n                               ! 循环地画出每个数据的直方
          SET  COLOR  ″red″
          BOX  AREA  i-0.4，i+0.4，0，value(i)   ! 画直方
          SET  COLOR  ″green″
          PLOT  TEXT，AT  i，value(i)+5:str$(value(i)，  ! 标记数值
     NEXT  i
END  SUB
```

10.4.2 动画设计

如果将 BOX 语句的保存图象和重显图象的功能适当地组合运用，则可实现动画设计。显示物体的动态图象时，可以把物体的运动过程分解为一组对应于不同时刻的静态图象。只要按顺序将不同时刻的静态图象快速地显示出来，便能够产生动画效果。

显示物体某一时刻静态图象的过程为：

(1) 沿运动轨迹定位；(2) 保存该位置上的背景图象；(3) 在该位置上显示物体这一瞬间的静态图象；(4) 将物体的瞬间图象抹去，恢复背景图象。显示动画即循环地执行这一过程。

例 10 显示一个小球沿着正弦曲线作往复运动。

程序：

```
SET MODE "graphics"
SET WINDOW -1, 2*pi+1, -1.5, 1.5
SET COLOR "green"                                   ! 设置前景色为绿色
FOR x=0 to 2*pi step 0.02                           ! 用绿色画正弦曲线
    PLOT LINES:x, sin(x);
NEXT x
SET COLOR "yellow"                                  ! 前景为黄色
BOX CIRCLE 2*pi, 2*pi+0.3, 1, 1.2                   ! 在屏幕右上角画小球外圆
SET COLOR "red"                                     ! 设置前景色为红色
FLOOD 2*pi+0.1, 1.1                                 ! 用前景色填满小球
BOX KEEP 2*pi, 2*pi+0.3, 1, 1.2 IN ball$            ! 将小球图象存入 ball$
BOX CLEAR 2*pi, 2*pi+0.3, 1, 1.2                    ! 抹去屏幕上的小球
LET a=0
LET b=2*pi
LET c=0.05
DO                                                  ! 循环地显示小球的往返运动
   FOR x=a to b step c
       LET y=sin(x)                                 ! 沿轨迹定位
       BOX KEEP x-0.2, x+0.2, y-0.2, y+0.2 IN old$  ! 保存背景
       BOX SHOW ball$ AT x-0.15, y-0.1              ! 显示小球
       PAUSE 0.01                                   ! 延时 0.01 秒
       BOX SHOW old$ AT x-0.2, y-0.2                ! 恢复背景
       IF key input then STOP                       ! 按任意键则终止程序
   NXET x
   ! 改变运动方向:互换起点和终点，步长反向
   LET t=a
   LET a=b
   LET b=t
   LET c=-c
LOOP
END
```

*10.5 图画

为了提高绘图程序的设计效率及灵活性；可以将具有图形绘制功能的语句序列定义成一个子程序．此外，True BASIC 还允许将绘画语句序列定义成另一种形式的程序模块——图画．

图画是专用于绘图的程序模块．图画与绘图子程序的区别在于：调用图画时，除了可以代入不同的参数调整所绘图形外，还可以通过指定变换方式，对图形作平移、放大、旋转等几何变换．因此使用图画更便于灵活地绘图．

10.5.1 图画的定义和调用

图画的定义格式:

```
PICTURE  PicName(形式参数 1，形式参数 2，…)
              ……   ┐
              ……   │
              PLOT……  ├ (语句块)
              ……   │
              ……   ┘
END  PICTURE
```

其中，PicName 为图画名，可以按照变量的命名规则任意给定。形式参数表示绘图时需从外部代入的数据。

图画调用语句格式:

```
DRAW  PicName(实际参数 1，实际参数 2，  …)
```

作用：调用名为 PicName 的图画.

例 11 画一组以原点为中心的正方形。

首先设计一个名为 square 的图画，画出一个关于原点对称的正方形，正方形的边长数据以及颜色编号由外部代入。然后编制主程序，以不同的参数对图画 square 进行多次调用.

程序:

```
PICTURE  square(s, c)
    SET  COLOR  c
    PLOT  LINES: -s,  s;  s,  s;
    PLOT  LINES: s,  -s;  -s,  -s;
    PLOT  LINES: -s,  s
END  PICTURE
!
SET  MODE  "Graphics"
SET  WINDOW  -16,  16,  -10,  10          ! 以屏幕中心作为坐标原点 (0，0)
FOR  i = 1 to 10
     LET  ColorNo = mod(i,  3)+1
     DRAW  square(i,  ColorNo)
NEXT  i
END
```

运行这个程序，屏幕上将显示十个颜色相间的矩形，当窗口参数的设置符合屏幕纵横比时，便可显示成正方形.

在图画中可以包含 EXIT PICTURE 语句. 该语句被执行时，将退出图画并返回到主程序. 类似于子程序，一个图画可以调用其他图画，甚至递归地调用自身. 图画也分内部图画，外部图画，并可置于库中.

10.5.2 图画的几何变换

True BASIC 允许在调用图画时对所绘图形进行各种几何变换. True BASIC 提供了常用的基本变换。各种复杂的几何变换则可由基本变换的组合来实现。此外，用户也可根据需要定

义特殊的变换。

True BASIC 提供了 5 种基本变换。

SHIFT(a，b) —— 平移变换。将图形平移到另一位置。图形上每一点 (x，y)被移到 (x+a，y+b)。

SCALE(a) ---- 变比变换。改变图形的大小。图形上每一点 (x，y)的坐标值都乘以因子 a，变成(x * a，y * a)。当 a>1 时图形被放大，当 a<1 图形被缩小。x 轴方向和 y 轴方向上的变化比例相同。

SCALE(a，b) —— 变比变换。改变图形的大小。图形上每一点 (x，y)的坐标值分别乘以 a，b，变成 (x * a，y * b)。x 轴方向和 y 方向上的变化比例不相同。

ROTATE(a) —— 旋转变换。将图形沿逆时针方向绕坐标原点旋转a弧度。当使用 OPTION ANGLE DEGREES 语句说明后，a 则为角度。

SHEAR(a) —— 错移变换。将图形中的所有垂线向前(沿顺时针方向)倾斜 a弧度。图形上每一点(x，y)将变成 (x+y * tan(a)，y)。当使用 OPTION ANGLE DEGREES 语句说明后，a 则为角度。

如果调用图画进行绘图时需要对所绘图形作变换，则可用带 WITH 的 DRAW 语句。

语句格式：

```
DRAW  PicName(实际参数 1，实际参数 2，…)WITH  trans
```

作用：调用名为 PicName 的图画，并按指定的变换方式 trans 对所绘图形进行几何变换。

例如，调用图画 square 时，除代入边长参数外，还可对其进行平移或旋转：

```
DRAW  square(1,1)  WITH  shift(3, 3)          (平移)
DRAW  square(1,2)  WITH  rotate(pi/4)         (旋转)
```

例 12 绘制由树木和房屋构成的风景画。

程序定义了图画 tree 和 house，分别在坐标原点上绘出树木和房屋。主程序反复调用这两个图画，并利用平移变换 SHIFT 将图形移到指定的位置上。由于图画的平移位置是随机产生的，因此每次运行都可得到不同的图案。

程序：

```
PICTURE  tree                                           !定义画树的图画
        SET  COLOR  ″green″                             !用绿色画树叶
        PLOT  AREA:-0.3, 0.2; 0.3, 0.2; 0, 0.5          !画下层树叶
        PLOT  AREA:-0.2, 0.4; 0.2, 0.4; 0, 0.8          !画中层树叶
        PLOT  AREA:-0.14, 0.7; 0.14, 0.7; 0, 1          !画上层树叶
        SET  COLOR  ″yellow″                            !用黄色画树杆
        PLOT  LINES:0, 0; 0, 0.2                        !画树杆
END  PICTURE
!
PICTURE  house                                          !定义画房子的图画
        SET  COLOR  ″yellow″                            !用黄色画墙和烟囱
        PLOT  AREA:-0.5, 0; 0.5, 0; 0.5, 0.6; -0.5, 0.6 !画墙
        PLOT  AREA:0.3, 0; 0.4, 0; 0.4, 1.1; 0.3, 1.1   !画烟囱
        SET  COLOR  "red"                                      ! 用红色画屋顶门窗
        PLOT  AREA:-0.6, 0.6; 0.6, 0.6; 0, 0.9                 ! 画屋顶
        PLOT  AREA:-0.4, 0; -0.2, 0; -0.2, 0.4; -0.4, 0.4      ! 画门
        PLOT  AREA:0.1, 0.2; 0.4, 0.2; 0.4, 0.4; 0.1, 0.4      ! 画窗
   END  PICTURE
   !
   SET  MODE  "graphics"
   SET  WINDOW  -5, 5, -3, 3
   RANDOMIZE
   FOR  y=2  to  -3  step  -0.5                           ! 从上到下(即从远到近)按行画树
        FOR  i=1 to 5                                     ! 每行画 5 棵树
             LET  x=8*rnd-4                               ! 树的左右位置由随机函数决定
             DRAW  tree  with  shift(x, y)                ! 画树
             LET  j=j+1
             IF  mod(j, 14)=0  then                       ! 每画 14 棵树画一所房屋
                DRAW  house  with  shift(x+0.5, y)    。
             END  IF
        NEXT  i
   NEXT  y
   END
```

除使用上述基本变换外，还可以将基本变换组合起来，构成新的几何变换。比如，绕任意点 (x, y) 旋转的几何变换可通过顺序执行 shift(-x, -y), rotate(a), shift(x, y)基本变换来实现。

对图画进行组合变换时，如果所作的变换由基本变换 trans1, trans2, …构成，那么 **DRAW** 语句 **WITH** 后的变换方式 trans 应写成 trans1 * trans2 * …的连乘形式。

例 13 将一个动物图案旋转。

由于定义动物图案时原点定在图案的左下角，因此直接绕原点旋转将造成图案互相重叠。为了避免重叠，程序中先将图案平移，使其离开原点，然后再将平移后的图形绕原点旋转(图 10-5)。

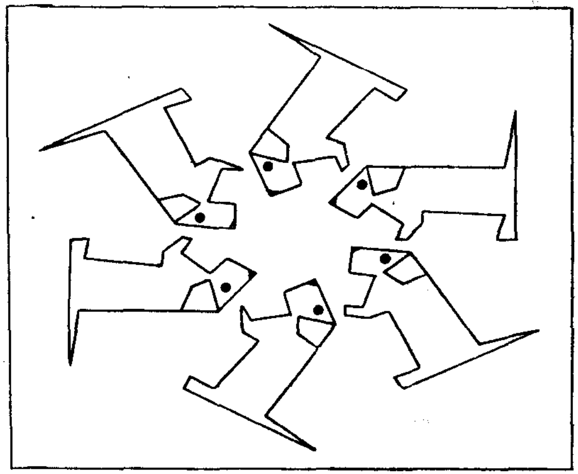


图 10-5　小狗图案的旋转

程序:

```
   PICTURE  dog                                   ! 定义画小狗图案的图画
        SET  COLOR  "yellow"
        PLOT  LINES: 0.35, 0; 0.4, 0.1; 0.25, 0.25; 0.15, 0.15; 0, 0.25;
        PLOT  LINES: 0.2, 0.5; 0.95, 0.5; 1, 0.85; 1, 0; 0.9, 0; 0.93, 0.15;
        PLOT  LINES: 0.5, 0.2; 0.5, 0.1; 0.4, 0; 0.35, 0
        PLOT  LINES: 0.2, 0.5; 0.25, 0.3; 0.35, 0.35; 0.4, 0.5
        SET  COLOR  "red"
        PLOT  AREA: 0.15, 0.35; 0.15, 0.38; 0.18, 0.38; 0.18, 0.35; 0.15, 0.35
        PLOT  AREA: 0, 0.25; 0.04, 0.3; 0.03, 0.23; 0, 0.25
    END  PICTURE
    !
    SET  MODE  "Graphics"
    SET  WINDOW  -2, 2, -1.4, 1.4
    FOR  i=0 to 5                    ! 画 6 次小狗图案，每次较前次旋转 π/3 弧度
        DRAW  dog  with  shift(0.2, 0) * rotate(i * pi / 3)
    NEXT  i
    END
```

注意：组合变换中各基本变换的先后顺序一般是不能交换的。例如先平移再旋转与先旋转再平移的效果并不相同。因此 shift(x, y) * rotate(a)与 rotate(a) * shift(x, y)不能相互代替。

另外，当对图画进行变换时，变换仅作用于由 PLOT 语句产生的图形，而由 BOX 语句产生的图形和 FLOOD 语句着色点的位置将保持不变。因此，在需要进行变换的图画中一般不使用 BOX 和 FLOOD 语句。

*10.6 坐标输入、单键输入和多重窗口

能够对图形的形状、位置、色彩等因素进行动态控制的绘图过程称为交互式绘图。计算机辅助设计(CAD)和计算机游戏便是交互式绘图的典型例子。实现交互式绘图需要计算机在绘图过程中能够方便灵活地接收用户的操作信息。True BASIC 提供了坐标输入和单键输入两种灵活的信息输入方式。坐标输入可以读取屏幕上任意点的坐标值，直接用于作图；单键输入可以在程序运行时读取键盘上某个键的代码，可用于控制绘图过程。运用多重窗口技术可以在屏幕上分区显示不同的图形或正文。开设多重窗口，能够同时独立地显示多种信息，并利于对信息进行相互对照。

10.6.1 坐标输入

语句格式：

GET POINT: x, y

作用：在屏幕上产生一图形光标(+)，此光标可在“光标键”的控制下移动。若输入回车键，则将当前图形光标所在点的横纵坐标值分别读入变量 x 和 y 中。

移动光标时，键盘上各“光标键”的作用如下：

↑，↓，←，→ ——将图形光标上，下，左，右移动。
Home，End ——将图形光标移至左上角，右下角。
PgUp，PgDn ——将图形光标快速上移，快速下移。
⇤ ，⇥ ——将图形光标快速左移，快速右移。
Ctrl←，Ctrl→ ——将图形光标移至屏幕左侧，右侧。

例 14 在屏幕上手控图形光标画图。

程序运行时，屏幕上显示出一个作图区域，并出现一个图形光标，可移动光标画折线。

图形由一组折线构成。

画折线时，先将光标移到折线的起点位置上，并按回车键表示折线开始。然后将光标依次移到折线的各点，每移至一点按一次回车键。此时在该点与上一点之间便画出一直线段，使折线延续构成图形。如果需要终止折线，则在画至终点后，在终点位置上再按一次回车键。图形画完之后，将光标移到作图区域外按回车键，程序即可结束。

程序:

```
SET  mode  "graphics"
SET  window  -11, 11, -11, 11
BOX  LINES  -10, 10, -10, 10                    ！显示作图区域
DO
     GET  POINT:X, Y                            ！移动光标并读入坐标值
     IF  abs(x)>10  or  abs(y)>10  then  EXIT  DO  ！在区域外则程序结束
     IF  x=x0  and  y=y0  then                  ！位置不变则终止折线
         PLOT  LINES
     ELSE                                       ！否则与上一点相连
         ！重复同一坐标可以画出起点
         PLOT  lines: x , y ; x, y;
     END IF
     LET  x0=x                                  ！保存这点的坐标值
     LET  y0=y
LOOP
END
```

例 15　图案设计。

本程序运行时，把整个屏幕作为制图区域。开始时在屏幕的左上角显示出两个事先设计好的基本图形(树木和房屋)，在屏幕中心出现图形光标。利用这两个基本图形，可以在屏幕上组合出较复杂的图案。

进行屏幕图案设计时，只需将图形光标移入基本图形所在的方框之内，并按回车键，该图形便被选定。此时移动光标并按回车键，被选定的图形便复制到指定位置上。基本图形一经选定，即可反复使用，直至改选另一基本图形为止。将光标移到图框之外并按回车键，则程序运行结束。

程序:

```
(将例 12 程序中的 PICTURE tree 及 PICTURE house 放在 PICTURE object 之前)
     ......
     ......
PICTURE  object(i)           ！将基本图形归入object并进行编号
     IF  i=1  then           ！tree的编号为1
         DRAW  tree
     ELSE  IF  i=2  then     ！house的编号为2
         DRAW  house
     END  IF
END  PICTURE
```

```
! 主程序
SET  MODE  "graphics"
SET  WINDOW  -5.1, 5.1, -3.1, 3.1
BOX  LINES  -5, 5, -3, 3                                   ! 制图区域
BOX  LINES  -5, -4, 2, 3                                   ! 第一个基本图形方框
DRAW  object(1)  with  scale(0.8)*shift(-4.5, 2)           ! 方框内画出基本图形
BOX  LINES  -5, -4, 1, 2                                   ! 第二个基本图形方框
DRAW  object(2)  with  scale(0.8)*shift(-4.5, 1)           ! 方框内画出基本图形
DO
     GET POINT:x, y                                        ! 读光标位置
     IF  abs(x)>5  or  abs(y)>3  then  EXIT  DO            ! 在制图区域外则退出
     IF  x<=-4  and  y>=2  then                            ! 在第一个基本图形方
         LET  k=1                                          ! 框内则选 1 号图形
     ELSE  IF  x<=-4  and  y>=1  and  y<2  then            ! 在第二个基本图形方
         LET  k=2                                          ! 框内则选 2 号图形
     ELSE                                                  ! 否则画出已选图形
         DRAW  object(k)  with  shift(x, y)
     END  IF
LOOP
END
```

10.6.2　单键输入

语句格式:

GET　KEY　a

作用: 等待键盘输入，当键盘上输入某个键后，将这个键的代码值赋给变量 a。

以下程序可以显示出键盘上任意键的代码:

```
DO                                         ----------------------
     PRINT  "Please type a key"            Ok.RUN
     GET  KEY  a                           Please type a key   (按↑键时)
     PRINT  "The code is"; a               The code is 328
LOOP                                       Please type a key   (按↓键时)
END                                        The code is 336
                                           ......
```

程序运行时，只要按下任一键(包括需要用 shift, ctrl 或 Alt 配合操作的键)，便在屏幕上显示出这个键的代码值。

类似于 INPUT 语句，程序执行 GET　KEY 语句时，计算机将一直等待键盘输入，直至按下任一键。如果不需等待，可通过检测条件 KEY　INPUT (检查用户是否按键，是为真，否为假)来判断键盘上是否有键输入，当条件成立时，才由 GET　KEY 语句读取其代码。

与 INPUT 语句不同的是，GET KEY 语句将键盘的按键直接输入，不需等待回车，因而操作反应快。另外， GET KEY 语句不会将键盘的按键显示出来，因此，用于控制作图时不影响屏幕上的图形。

例 16　用键盘控制图形的移动

屏幕上显示出棋盘和棋子，通过←、→、↑、↓键在棋盘上移动棋子。

程序:

```
SET MODE "graphics"
SET CURSOR "off"                                  ! 取消屏幕光标
SET WINDOW -2, 11, 0, 8
DRAW horse                                        ! 画棋子"马"
BOX KEEP 0, 1, 0, 1 IN horse$                     ! 保存"马"的图形
BOX CLEAR 0, 1, 0, 1
SET COLOR "yellow"
FOR x=0 to 8                                      ! 画棋盘
    PLOT x, 0; x, 8
    PLOT 0, x; 8, x
NEXT x
LET x, y=3                                        ! 确定"马"的初始位置
DO
    BOX SHOW horse$ AT x, y USING "xor"           ! 在规定位置上重显"马".
    GET KEY a                                     ! 读取键盘按键的代码
    BOX SHOW horse$ AT x, y USING "xor"           ! 消除"马"的图形
    SELECT CASE a                                 ! 根据代码确定移动方向
        CASE 328                                  ! 按↑键时向上移
            LET y=y+1
        CASE 331                                  ! 按←键时向左移
            LET x=x-1
        CASE 333                                  ! 按→键时向右移
            LET x=x+1
        CASE 336                                  ! 按↓键时向下移
            LET y=y-1
        CASE ELSE                                 ! 按其他键则程序结束
            EXIT DO
    END SELECT
    LET x=min(max(x, 0), 7)                       ! 将x限制在棋盘范围内
    LET y=min(max(y, 0), 7)                       ! 将y限制在棋盘范围内
LOOP
END
PICTURE horse                                     ! 定义棋子"马"的图形
    SET COLOR "green"
    PLOT .2, .1, .8, .1, .8, .2, .7, .25, .7, .3, .8, .4, .65, .7,
    PLOT .6, .9, .55, .9, .5, .82, .2, .75, .2, .6, .3, .6, .4, .55,
    PLOT .25, .45, .2, .37, .3, .3, .3, .25, .2, .2, .2, .1
    SET COLOR "red"
    FLOOD 0.5, 0.5
END PICTURE
```

10.6.3 多重窗口

True BASIC 允许同时在屏幕上开设多个独立窗口。每个窗口都有各自的窗口编号，其中，0 号窗口默认为整个屏幕。程序中未进行窗口设置时，True BASIC 自动使用 0 号窗口，程序产生的数据或图形显示在整个屏幕范围内。

此外，还可以把屏幕分为几个不同的矩形部分，每个部分设置成独立的窗口，以便将屏幕分开使用。

开设窗口语句格式：

OPEN #n: SCREEN left, right, bottom, top

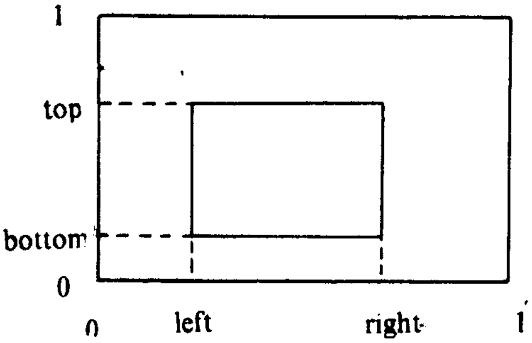


图 10-6 窗口矩形区域在屏幕上的位置

作用：开设 n 号窗口(#n 称为通道号，1<n<999)。

窗口在屏幕上的矩形区域是 [left<x<right; bottom<y<top]。用 1 表示屏幕的宽和高，则

0<left < right<1; 0<bottom < top<1

如图 10-6 所示。

开设了某一窗口之后，所有与显示有关的内容都将显示在这一窗口之内。此时由 SET WINDOW 语句设置的窗口坐标系也是针对所开设的窗口，而不是针对整个屏幕的。开设新的窗口不影响屏幕原有的显示内容。

例 17 分窗口显示多种玫瑰线。

程序将屏幕上下左右等分，开设 4 个窗口，每个窗口内显示一种玫瑰线的图形。

程序：

```
SET MODE "Hires"
OPEN#1:SCREEN 0, 0.5, 0.5, 1           ! 将 1 号窗口设置成左上方四分之一屏幕
CALL RoseCurve(2)                      ! 画玫瑰线 r=sin(2t)
OPEN#2:SCREEN 0.5, 1, 0.5, 1           ! 将 2 号窗口设置成右上方四分之一屏幕
CALL RoseCurve(3)                      ! 画玫瑰线 r=sin(3t)
OPEN#3:SCREEN 0, 0.5, 0, 0.5           ! 将 3 号窗口设置成左下方四分之一屏幕
CALL RoseCurve(4)                      ! 画玫瑰线 r=sin(4t)
OPEN#4:SCREEN 0.5, 1, 0, 0.5           ! 将 4 号窗口设置成右下方四分之一屏幕
CALL RoseCurve(5)                      ! 画玫瑰线 r=sin(5t)
END
!
SUB RoseCurve(n)                       ! 画玫瑰线 r=sin(nt)的子程序
    SET WINDOW -1.8, 1.8, -1.3, 1.3
    BOX LINES -1.7, 1.7, -1.25, 1.25
    DEF x(t)=sin(n * t) * cos(t)       ! 将极坐标方程转化成参数方程
    DEF y(t)=sin(n * t) * sin(t)
    FOR t=0 to 2 * pi step pi / 30
        PLOT LINES:x(t), y(t);
```

```
NEXT t
PLOT LINES
PLOT TEXT, AT 0.5, 1:"r=sin("&str$(n)&"t)"
END SUB
```

如果开设了多个窗口，当需要在不同窗口内轮流显示信息时，应当用 WINDOW 语句实行窗口转换.

语句格式:

WINDOW#n

作用: 将显示转换到第 n 号窗口。

转入一窗口时，将继续使用上次转离此窗口时的颜色、光标位置和窗口参数。

例 18 在屏幕上分区显示数据的曲线图形和数据的值。

程序共开设三个窗口。其中，1 号窗口用显示曲线图形；2 号窗口用来标注数据；3 号窗口套在 2 号窗口之内，用来显示数据的值。假设数据为 1960 到 1980 年每年的产品出口率。

程序:

```
SET MODE "graphics"
OPEN#1:SCREEN 0, 0.49, 0, 1
SET WINDOW 1960, 1980, 0, 100
BOX LINES 1960, 1980, 0, 100
SET COLOR "red"
!
OPEN#2:SCREEN 0.51, 1, 0, 1
BOX LINES 0, 1, 0, 1
SET CURSOR 3, 3
SET COLOR "green"
PRINT "Year Export(%)"
!
OPEN#3:SCREEN 0.53, 0.97, 0.11, 0.82
BOX LINES 0, 1, 0, 1
SET COLOR "yellow"
!
FOR year=1960 to 1980
    READ export
    WINDOW#1
    PLOT LINES:year, export;
    WINDOW#3
    PRINT year; tab(8); export
NEXT year
!
DATA 50, 72, 44, 68, 12, 34, 23, 45, 42, 50
DATA 64, 72, 79, 74, 80, 83, 82, 90, 92, 94, 97
END
```

注意：在窗口中显示图形时，如果图形超出窗口范围，则超出的部分将被忽略；在窗口中显示字符或数字时，如果一行中的字符或数字超出了窗口的宽度，则会引起显示混乱。为了避免出现这种情况，可用语句

SET♯n: MARGIN 行宽

设置n号窗口中显示字符或数字的行宽(每行的字符数)，使行宽小于等于该窗口中一行所能显示的最大字符数。

另外，不再向某个窗口输出显示内容时，可用语句

CLOSE♯n

将窗口关闭，此时通道号♯n可用于开设其他窗口。

10.7 声音与音乐

微型计算机在硬件配置上一般都设有一个简单扬声器，在程序控制下能够实现音响输出。True BASIC中设有两个与音响有关的语句：Sound语句和Play语句。

10.7.1 SOUND语句

语句格式：

SOUND 发声频率，发声持续时间

其中，频率以赫兹(Hz)为单位，时间以秒(s)为单位。

当SOUND语句被执行时，扬声器将按给出的持续时间发出指定频率的声音；如果紧接该SOUND语句的是另一SOUND语句，则后一SOUND语句需待前一SOUND语句发声持续时间结束后才开始发音.发声持续时间内不影响其他语句(SOUND除外)的同时执行。

例如，以下程序以秒为单位时间，每次报时5秒钟：

```
DO
   FOR  i=1 to 4
        SOUND  1000, 0.5
        SOUND  0, 0.5
   NEXT  i
   SOUND  1000, 2
   PAUSE  1                              ! 暂停一秒钟
   IF  key  input  then  EXIT  DO        ! 按任意键则退出
LOOP
END
```

以下程序用于模拟电话铃声：

```
FOR  i=1 to 8
     FOR  j=1 to 3
          SOUND  600, 0.2
          SOUND  1600, 0.2
     NEXT  j
     PAUSE  1
NEXT  i
END
```

例 19　利用↑、↓键对声调进行控制。

程序：

```
LET  x=600                            ! 初始频率设为600赫兹
DO
    IF  key  input  then
        GET  KEY  a
        SELECT  CASE  a
        CASE  328                     ! 按↑键时频率增高
              LET  x=x+20
        CASE  336                     ! 按↓键时频率降低
              LET  x=x-20
        CASE  ELSE                    ! 按其他键则程序停止
              STOP
        END  SELECT
    END  IF
    SOUND  x, 0.02
LOOP
END
```

10.7.2　PLAY 语句

语句格式：

PLAY 乐谱字符串

其中，乐谱字符串由一组 True BASIC 音乐命令构成.执行 PLAY 语句时，将按乐谱字符串的要求演奏出相应的乐曲。

例如，按四分音符演奏 C 调 1234567 七个音符的语句为：

PLAY ″L4 O4 CDEFGAB″

True BASIC 的音乐命令如下：

A 到 G	演奏一个音符。其中C，D，E，F，G，A，B分别对应于C调唱名的1，2，3，4，5，6，7。
♯，+和−	跟在音符之后，使音符升高(♯、+)或降低(−)半个音。例如 C+，G−。
O0 到 O7	设置当前的音阶。O0到O7分别对应于钢琴键盘上的甚低音组、低音组、大字组、小字组、小字一组、小字二组、小字三组和小字四组的音阶。中央 C 为 O5 的第一个音符 C。未设置音阶则默认音阶为 O4。
> 和 <	向上(>)或向下(<)移动一个音阶。例如O4 AB>CD中的CD位于O5音阶。
Ln	设置随后各音符的音长。整数n表示将音符设为n分音符。例如L8表示八分音符，L16 表示十六分音符。如果仅改变单个音符的音长，则可在该音符后标明音长，如 A2、B4。
•	跟在音符后构成附点音符。
Rn 或 Pn	休止符。整数n表示按n分音符休止。
Tn	设置演奏速度。整数n表示每分钟演奏n个四分音符。
MN	按正常方式演奏。每个音符演奏7／8音长，休止1／8音长，乐曲显得干净利落。

ML	按连奏方式演奏。每个音符均演奏全音长，乐曲显得缓慢连绵。
MS	按断奏方式演奏。每个音符演奏3／4音长，休止1／4音长，乐曲显得活泼跳跃。
MB	将音乐作为背景音乐演奏，演奏的同时允许程序继续运行。MB后最多可写32个音符，多出的音符将被忽略。程序结束时，背景音乐自动停止。
MF	将音乐作为前景音乐演奏，即演奏时不执行其他语句。

在乐谱字符串中，可任意使用空格，并且不区分字母的大小写。

例 20 演奏《天鹅湖》中的“四只小天鹅”。

程序:

```
LET a$ ="T100 O4 L8 MS <A"
LET b$ =">CCCMLC.<B32>C32MSDC <B>DDDMLD.C32D32MSED"
LET c$ ="CEAG+ ML E2 MS EEAG+ ML E2 MS E"
LET d$ ="MS EEEEE L16 ML AGFE L8 D MN E4F4C+4 D4 MS D"
LET e$ ="DDDD L16 ML GFED L8 C"
LET f$ ="MS G L16 ML BAGF L8 E MS B L16 ML >DC<BA L8 G"
LET g$ ="MS <A L16 ML B>CDD+ L8 E MS <E L16 ML FF+GG+A"
PLAY a$&b$&c$&b$&c$
PLAY "<"&d$&e$&f$&d$&e$&g$
END
```

*10.8 程序举例

例 21 绘制函数曲面的立体图形。

应用问题中有许多实际数据能够作为函数曲面来处理，例如高低起伏的地形、工业产品的外形等等。函数曲面是三维形体，直观表现三维形体的一般方法是绘制它的立体图形。本例介绍绘制单值函数曲面立体图形(图 10-7)的一种简单而基本的方法。

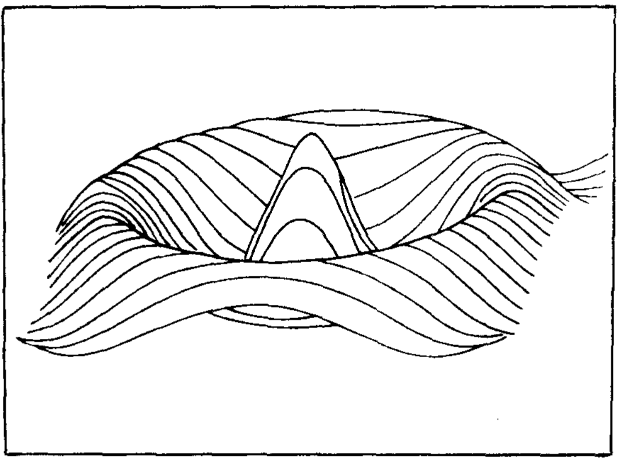


图 10-7 函数曲面立体图

为了绘制函数曲面的立体图形，需要进行以下三个方面的处理:

(1) 坐标换算 把曲面上的点在三维空间中的坐标(x, y, z)换算成绘图时屏幕上的坐标(u, v)。程序中从三维空间 XYZ 坐标系到屏幕 UV 坐标系的换算公式取:

$$u = x + y \times d \times \cos(\alpha)$$

$$v = y \times d \times \sin(\alpha) + z$$

其中，α 为 Y 轴在屏幕 UV 平面上与 U 轴的夹角($0 < \alpha < 360°$)，d 为 Y 轴的单位长度在 UV 平面上的实际长度($0 < d < 1$)。为了获得不同的立体效果，可选择不同的 α 和 d(图 10-8)。

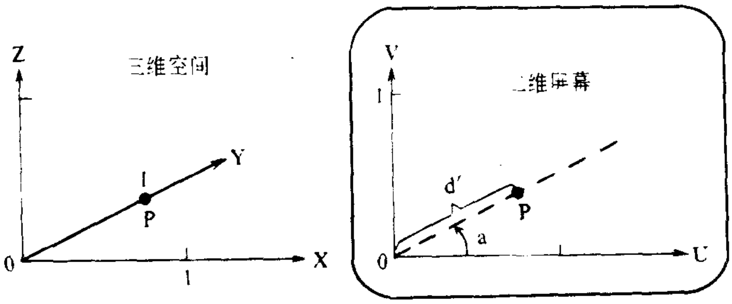


图 10-8　在维空间到屏幕的转换

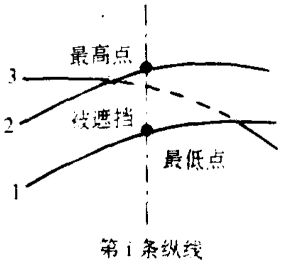


图 10-9　消隐示意

(2) 截线　即用由前到后的一组平面 $Y=C_i$ 与曲面相交，得到一组截线。画曲面实际上是由前到后依次画出这组截线。

(3) 消隐　为了表现立体效果，必须让前面的截线挡住后面的截线。为此使用两个数组 *h*，*l*，其中 *h*(*i*)，*l*(*i*)分别表示屏幕第 i 条纵线上被遮挡部分的最高点和最低点。当后面的截线进入这两点之间时，则因被遮挡而显示不出来（图 10-9)。

设曲面函数为

$$f(x,\ y)\ =\cos\ (5\sqrt{x^2+y^2}\ )\ /(1+\sqrt{x^2+y^2}\ )$$

程序：

```
DEF  f(x, y)                                  ! 定义曲面函数
     LET  t=sqr(x * x+y * y)
     LET  f=cos(5 * t)/(1+t)
END  DEF
!
SUB  ToScreen(x, y, z, u, v, i)               ! 坐标换算子程序
     LET  u=x+y * d * cos(a)                  ! 三维坐标(x, y, z)-->屏幕坐标(u, v)
     LET  v=y * d * sin(a)+z
     LET  i=(u-left) * 319/(right-left)+1     ! i 为(u, v)所在的屏幕纵线
END  SUB
!
LET  left, bottom=-2.3                        ! 屏幕的窗口范围
LET  right, top=2.3
LET  x0, y0=-1.5                              ! 曲面的作图范围
LET  x1, y1=1.5
!
DIM  h(320), l(320)                           ! 开辟消隐数组
MAT  h=(maxnum)                               ! 在绘图前, h 中所有元素均取极大值
!
INPUT  prompt "a=?": a                        ! 输入夹角 a 的角度值, 0<a<360°
LET  a=rad(a)                                 ! 将角度换算成弧度
INPUT  prompt "d=?": d                        ! 输入比例 d, |d|<1
!
```

```
SET MODE "graphics"
SET WINDOW left, right, bottom, top
FOR y=y0 to y1 step (y1-y0)/24                       ! 画 25 条截线
    FOR x=x0 to x1 step (x1-x0)/200                  ! 每条截线取 201 段
        CALL ToScreen(x, y, f(x, y), u, v, i)        ! 坐标变换
        IF i>=1 and i<=320 then                      ! 判断是否在屏幕范围内
           IF h(i)=maxnum then                       ! 如果是纵线 i 上的第一条截线
              PLOT LINES: u, v;                      ! 画出截线段
              LET h(i), l(i)=v                       ! 登记最高、最低消隐点
           ELSE IF v>h(i) then                       ! 否则如果高于最高消隐点
              PLOT LINES: u, v;                      ! 画出截线段
              LET h(i)=v                             ! 重新登记最高消隐点
           ELSE IF v<l(i) then                       ! 否则如果低于最低消隐点
              PLOT LINES: u, v;                      ! 画出截线段
              LET l(i)=v                             ! 重新登记最低消隐点
           ELSE IF round(i)<>round(i0) then          ! 否则如果前后端点不共纵线
              PLOT LINES                             ! 消隐截线段
           END IF
        END IF
        LET i0=i                                     ! 保留截线段端点的纵线位置
    NEXT x                                           ! 画截线的下一段
    PLOT LINES                                       ! 终止一条截线
NEXT y                                               ! 画下一条截线
END
```

```
----------------------------------------------------
Ok.RUN
α=? 70
d=? 0.65
```

程序在消隐截线时考虑了由于分辨度误差而引起错误消隐的情况：当截线中某一段太陡（斜率的绝对值太大）时，前后两点会落在屏幕的同一纵线上。这时可能使后一点进入该纵线的被遮挡部分而画不出来，导致截线的陡峭部分出现许多间断。为了避免出现这种情况，程序在消隐某一段截线时先作判断，只有前后两点不在同一纵线上时才消隐。

例 22 动画：一个小球在矩形区域内连续滚动。

小球的滚动方向可用 8 个光标键控制。若遇到矩形的边框，小球会自动反弹。当输入其他键时，程序运行结束。

程序：

```
SET MODE "Graphics"
SET WINDOW -3, 3, -2, 2
! 画出黄色小球图形，并将小球图形存入 ball$ 中
SET COLOR "Yellow"
```

```
BOX  CIRCLE  x-0.1，x+0.1，y-0.1，y+0.1
BOX  KEEP  x-0.1，x+0.1，y-0.1，y+0.1 IN ball$
BOX  CLEAR  x-0.1，x+0.1，y-0.1，y+0.1
！画作为屏幕背景的红色网格
SET  COLOR  ″Red″
FOR  x=-3 to 3 step 0.2
     PLOT  LINES：x，-2；x，2
NEXT  x
FOR  y=-2 to 2 step 0.2
     PLOT LINES：-3，y；3，y
NEXT  y
LET  x，y=0                                        ！小球的初始位置
DO
   ！第一次按异或方式重显小球，小球叠加在背景网格上
   BOX  SHOW  ball$  AT  x-0.1，y-0.1  USING  ″xor″
   LET  x0=x
   LET  y0=y
   IF  key  input  then
       GET  KEY  k
       SELECT  CASE  k
       CASE  327                                    ！按 Home 键
            LET  xstep=-0.05
            LET  ystep=0.05
       CASE  328                                    ！按↑键
            LET  xstep=0
            LET  ystep=0.05
       CASE  329                                    ！按 PgUp 键
            LET  xstep，ystep=0.05
       CASE  331                                    ！按←键
            LET  xstep=-0.05
            LET  ystep=0
       CASE  333                                    ！按→键
            LET  xstep=0.05
            LET  ystep=0
       CASE  335                                    ！按 End 键
            LET  xstep，ystep=-0.05
       CASE  336                                    ！按↓键
            LET  xstep=0
            LET  ystep=-0.05
       CASE  337                                    ！按 PgDn 键
            LET  xstep=0.05
            LET  ystep=-0.05
```

```
        CASE  ELSE                          ! 按其他键
              STOP
        END  SELECT
    END  IF
    ! 小球的位置(x，y)按步长 xstep 和 ystep 变化
    LET  x=x+xstep
    LET  y=y+ystep
    ! 如果遇到左右边框则改变 x 的变化方向
    IF  abs(x)>2.9  then
        LET  x=x0
        LET  y=y0
        LET  xstep=-xstep
        SOUND  400, 0.01
    END  IF
    ! 如果遇到上下边框则改变 y 的变化方向
    IF  abs(y)>1.9  then
        LET  x=x0
        LET  y=y0
        LET  ystep=-ystep
        SOUND  600, 0.01
    END  IF
    ! 第二次按异或方式重显小球，小球在背景网格上消失
    BOX  SHOW  ball$  AT  x0-0.1, y0-0.1  USING  "xor"
LOOP
END
```

例 23 绘制平面数值分布图。

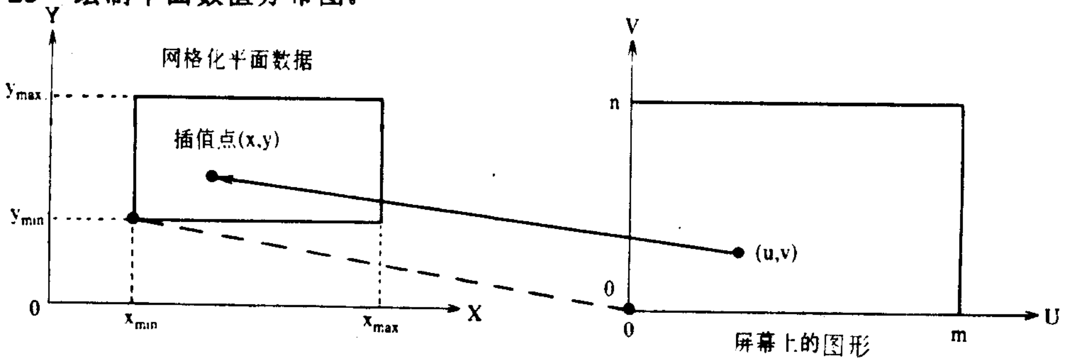


图 10-10 屏幕图形与平面数据的插值对应关系

在许多应用问题中，如地形、资源分布、摄影图片等，研究或处理的对象通常被表示成平面数据的形式．如果将平面上属于不同数值范围的数据用不同颜色的点来表示．那么便可以通过图形的色彩分布，直观地表现平面数据的数值分布情况．

本程序可以使用任意的网格化平面数据．网格化平面数据置于数组 a(0: r, 0: c)内，其中 r+1，c+1 分别为 x 方向和 y 方向上的数据个数．在屏幕上，以(m+1)×(n+1)的点阵显示出平面数据的数值分布彩色图形．(m+1)×(n+1)彩色图形中每点(u，v)所对应的数值，都是

在网格化平面数据上经插值得到的（图 10-10）。如果平面数据 x 方向和 y 方向的范围为 $[x_{min}, x_{max}; y_{min}, y_{max}]$，那么相应的插值点$(x, y)$为

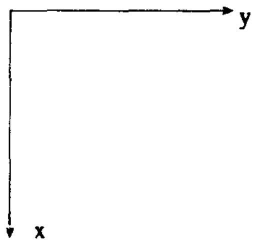
$$x = u / m \times (x_{max} - x_{min}) + x_{min}$$

$$y = v / n \times (y_{max} - y_{min}) + y_{min}$$

插值时调用库 insert.lib 中的曲面插值函数 f（参见第 9 章程序举例）。

程序：

```
LIBRARY "insert.lib"
DECLARE DEF f
! 读取网格化平面数据及平面的范围
DIM a(1, 1)
READ r, c, xmin, xmax, ymin, ymax
MAT READ a(0: r, 0: c)
!
DATA 5, 5                           ! x, y 方向上的数据个数减 1
DATA 1, 6, 1, 6                     ! xmin, xmax, ymin, ymax
!
DATA 2, 6, 5, 4, 7, 8                  !
DATA 3, 4, 4, 4, 5, 7                  !
DATA 5, 4, 4, 3, 4, 6                  !
DATA 5, 4, 3, 3, 3, 4                  !
DATA 4, 3, 3, 2, 2, 3                  !
DATA 5, 4, 2, 1, 0, 2                  !
```

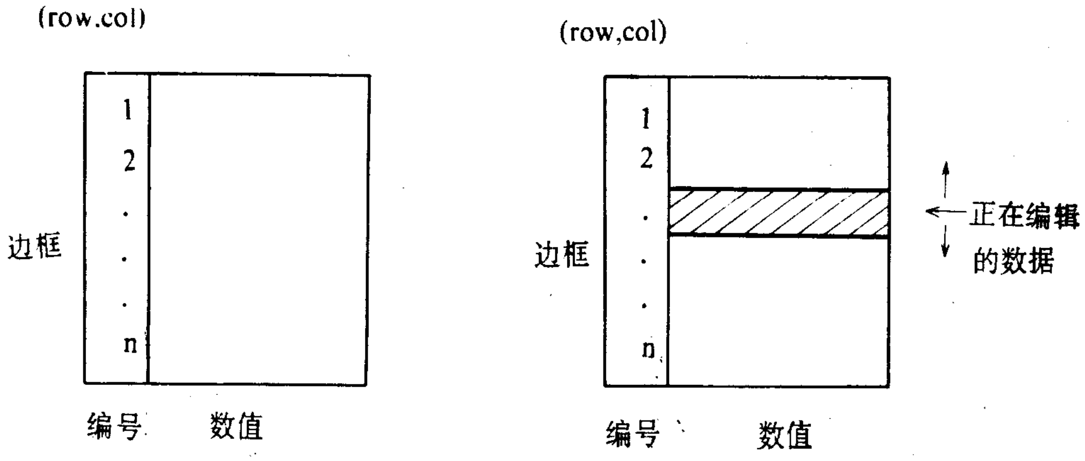


```
! 确定屏幕图形点阵的行列数
LET m, n = 70
! 计算窗口参数，使图形显示在屏幕中央
LET left = -159+int(m / 2)
LET right = 319+left
LET bottom = -99+int(n / 2)
LET top = 199+bottom
! 设置窗口并画出图形边框
SET MODE "graphics"
SET WINDOW left, right, bottom, top
SET BACK "blue"
SET COLOR "yellow"
BOX LINES -1, m+1, -1, n+1
! 定义屏幕图形中任一点(u, v)到网格化平面数据插值点(x, y)的换算公式
DEF x = u / m * (xmax-xmin)+xmin
DEF y = v / n * (ymax-ymin)+ymin
! 循环画出屏幕图形中的所有点
FOR u = 0 to m
    FOR v = 0 to n
```

```
        LET z=f(a( , ), xmin, xmax, ymin, ymax, x, y)    ! 计算插值点上的函数值
        IF  z<=1  then
            SET  COLOR  0                                ! 蓝色(当前背景色)
        ELSE  IF  z<=2  then
            SET  COLOR  mod(u+v, 2)                      ! 绿蓝相间
        ELSE  IF  z<=3  then
            SET  COLOR  1                                ! 绿色
        ELSE  IF  z<=4  then
            SET  COLOR mod(u+v,2)+3*mod(u+v+1,2)         ! 绿黄相间
        ELSE  IF  z<=5  then
            SET  COLOR  3                                ! 黄色
        ELSE  IF  z<=6  then
            SET  COLOR  3*mod(u+v, 2)+2*mod(u+v+1, 2)    ! 黄红相间
        ELSE
            SET  COLOR  2                                ! 红色
        END  IF
        PLOT  POINTS: u, v
    NEXT  v
NEXT  u
END
```

由于“graphics”图形方式下的颜色数目有限，因此用一种颜色表示一种数值等级的简单方法，对需要区分多个等级的情形是不适用的。为了解决这个问题，程序中通过将两种不同颜色的点交错排列的方法，构成新的色块，从而扩大了色块的种类。

例 24 以彩色表格的方式在屏幕上定位显示和随机编辑数组中的数据。



(a) ShowData　　(b) EditData

图 10-11　屏幕效果

本例运用了正文模式下有关颜色、光标定位、特殊字符等显示功能，使数组数据的输入输出具有良好的屏幕效果。显示数组时，能够自动将数组中的数据排列成彩色表格，并根据指定的位置将表格显示在屏幕上；编辑数组时，能够按照填写屏幕表格的方式输入或修改数

组中的任何一个数据。

例中通过两个子程序 ShowData 和 EditData，分别实现显示数组和编辑数组的功能。这两个子程序的形式参数为：

ShowData(v( )，row，col，n)

EditData(v( )，row，col，n)

其中，v( ) 为需显示或编辑的一维数值数组；row，col 为表格的左上角在屏幕上的行、列位置；n 为数据个数。调用子程序时的屏幕效果如图 10-11。

表格边框为黑底白色，数据编号为黑底绿色，数据内容为白底红色(奇数编号)或浅兰底红色(偶数编号)。

执行 EditData 时可以用光标键↑↓上下选择数据。被选中的数据为醒目的红底浅兰色，可对其进行输入或修改。执行完子程序 EditData 后须按 ESC 键退出。

子程序 ShowData 和 EditData 置于库 viewdata.lib 中：

程序：

```
EXTERNAL
!按表格方式显示数组的子程序
SUB ShowData(v(), row, col, n)
    !以黑底白色显示表格边框
    !边框符号"┌，─，┐，│，└，┘"的ASCII码为218，196，191，179，192，217
    SET COLOR "white/black"
    PRINT tab(row, col); chr$(218); repeat$(chr$(196), 21); chr$(191);
    FOR i=1 to n
         PRINT tab(row+i, col); chr$(179); repeat$(" ", 21); chr$(179);
    NEXT i
    PRINT tab(row+n+1, col); chr$(192); repeat$(chr$(196), 21); chr$(217);
    IF n=0 then
       PRINT tab(row, col+7); "No Data !"     !提示"无数据"
    ELSE
        FOR i=1 to n          !调用 ShowLine 显示表格中各行(编号和数据)
            CALL ShowLine(v( ), row, col, i)
        NEXT i
    END IF
END SUB
!按表格方式输入或修改数组数据的子程序
SUB EditData(v( ), row, col, n)
    CALL ShowData(v( ), row, col, n)                 !首先调用 ShowData 显示数组
    LET i=1                                          !i 为可编辑数据的下标，初值为 1
    IF n=0 then GET KEY w
    DO while n>0                                     !n>0 时循环编辑数组中的数据
        SET COLOR "cyan/red"                         !以红底浅兰色单独显示第 i 个数据
        PRINT tab(row+i, col+5); repeat$(" ", 17); tab(row+i, col+5); v(i)
        GET KEY a                                 !从键盘上接收一个键
        SELECT CASE a
        CASE 328                                  ! ↑键
            CALL ShowLine(v( ), row, col, i)      ! 将第i行恢复正常显示
            LET i=i-1                             ! i上移
        CASE 13, 336                              ! 回车键或↓键
            CALL ShowLine(v( ), row, col, i)      ! 将第i行恢复正常显示
            LET i=i+1                             ! i下移
        CASE 27                                   ! ESC键
            CALL ShowLine(v( ), row, col, i)      ! 将第i行恢复正常显示
            EXIT SUB                              ! 退出
        CASE ELSE                                 ! 其他键
            CALL InputData(row+i,col+5,v(i),a)    ! 键盘输入v(i)的新值
            CALL ShowLine(v( ), row, col, i)      ! 将第i行恢复正常显示
            IF a=328 then LET i=i-1               ! 按↑键结束输入则i上移
            IF a=13 or a=336 then LET i=i+1       ! 按回车键或↓键则i下移
            IF a=27 then EXIT SUB                 ! 按ESC键结束输入则退出
        END SELECT
        LET i=mod(i-1, n)+1                       ! i在1到n的范围内循环
    LOOP
END SUB
! 显示表格第i行(编号和数据)的子程序
SUB ShowLine(v( ), row, col, i)
    SET COLOR "green/black"                  ! 以黑底绿字显示编号
    PRINT tab(row+i, col+1);
    PRINT using "##": i
    IF mod(i, 2)=0 then
        SET COLOR "red/cyan"                 ! 以白底红色显示奇数行
    ELSE
        SET COLOR "red/white"                !以浅兰底红色显示偶数行
    END IF
    PRINT tab(row+i, col+5); repeat$(" ", 17); tab(row+i, col+5); v(i)
END SUB
! 在屏幕x行y列上定位输入变量v的数值，并具有容错功能的子程序
SUB InputData(x, y, v, a)                 ! a传入起始键和传出结束键的代码
    LET p$=""                             ! p$存放所按各键
    DO
        IF a=8 then                       ! 若按键是删除键(←)
        LET p$=p$[1:len(p$)-1]            ! 截去p$最右侧的字符

        ELSE IF pos("+-.1234567890eE", chr$(a))>0 then
        ! 若按键是正号、负号、小数点、数字及字母e和E
        LET  p$ =p$&chr$(a)                 ! 将该键加入 p$ 中
        ELSE                                ! 否则
            SOUND  600, 0.05                ! 放弃该键并发音警告
        END  IF
        PRINT  tab(x, y); p$&″ ″;           ! 显示 p$ 的内容
        GET  KEY  a                         ! 再接收一键
    LOOP  until  a>128  or  a=13  or  a=27
    ! 按回车键(13)、ESC 键(27)或代码大于 128 的键时结束输入
    ! 并将 p$ 中的字符内容转换成 v 的数值
    IF  p$ =″ ″then  LET  p$ =″0″
    WHEN  ERROR  IN                         ! 采用出错处理方法确保 v 为合法数值
          LET  v=val(p$)
    USE
          SOUND  600, 0.05
          LET  v, a=0
    END  WHEN
END  SUB
---------------------------------------------------------------------
```

```
Ok.SAVE  viewdata.lib
Ok.
```

为了避免出错信息破坏屏幕效果，EditData 中没有使用 INPUT 语句，而是通过子程序 InputData 为变量输数。用 InputData 输数时只接收与数字有关的键，并当输入内容不符合数字格式时会自动处理成 0，因此可以将操作错误屏蔽掉。

以下程序利用 viewdata.lib 中的 ShowData 和 EditData，对 10 个数进行排序。

程序：

```
LIBRARY  ″viewdata.lib″
SET  MODE  ″80″
SET  CURSOR  ″off″
DIM  v(10)
! 在屏幕左上角(1, 1)按表格方式为数组 v 输入 10 个数据
CALL  EditData(v( ), 1, 1, 10)
! 对数组 v 中的 10 个数据进行排序
FOR  i=1  to  9
     FOR  j=i+1  to  10
          IF  v(i)>v(j)  then
              LET  t=v(i)
              LET  v(i)=v(j)
              LET  v(j)=t
          END  IF
     NEXT  j
NEXT  i
```

```
! 在屏幕右上角(1, 54)按表格方式显示 v 中的 10 个数据
CALL  ShowData(v( ), 1, 54, 10)
END
```

***例 25**　菜单驱动程序的设计。

菜单驱动是指程序运行时屏幕上显示出代表程序各项功能的图形或文字（即菜单）。只要通过某种简单的方式（比如使用光标键或鼠标器）选定其中的某个图形或文字（即选菜单），程序便会自动执行相应的功能。菜单驱动方式使程序功能直观清晰，操作方法简便灵活，因此近年来得到越来越广泛的运用。

下面讨论用于显示和选取菜单的子程序的设计方法。

(1) **菜单子程序的设计**　利用光标键进行选项的文字菜单主要有横移式菜单和下拉式菜单两种形式(图 10-12)。

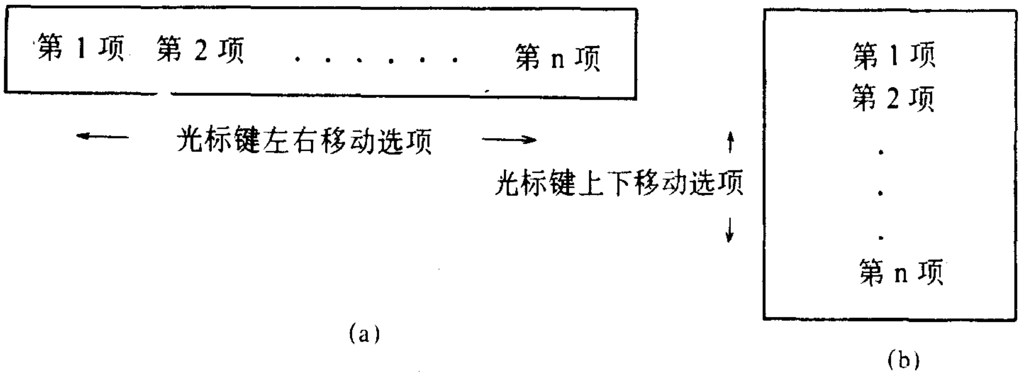


图 10-12　文字菜单形式

(a) 横移式菜单　　(b) 下拉式菜单

不论是哪一种形式的菜单，显示和选取菜单的子程序均可使用以下算法：

```
显示边框
显示各个项目
反复循环：
    用醒目的颜色突出显示第 i 项
    从键盘上接收一个键，然后判断
        →或↓键：i 加 1，令选取标志为-1，表示越过此项
        ←或↑键：i 减 1，令选取标志为-1，表示越过此项
          回车键：选中此项，令选取标志为 i 并退出循环
          ESC 键：不选任何项，令选取标志为 0 并退出循环
    将原来突出显示的项恢复正常显示
```

根据这一算法编制的横移式菜单子程序 menu1 固定显示在屏幕上方的第一行；下拉式菜单子程序 menu2 则可以定位显示在屏幕的任何位置上。这两个子程序的形式参数如下：

memu1(n,mn$,choice)--	n	菜单项目的个数
	mn$	菜单项目的名称(项目之间隔以空格)
	choice	选取标志
menu2(row,col,n,l,mn$,choice)--	row, col	菜单左上角在屏幕上的行，列位置
	n, l	菜单项目个数及菜单宽度(字符数)
	mn$	菜单项目的名称(项目之间隔以空格)
	choice	选取标志

这两个子程序按如下形式置于库 menu.lib 中:

```
EXTERNAL
SUB  menu1(n, mn$, choice)
     ! 调用辅助子程序 ToArray, 将 mn$ 中的 n 个项目名称拆开并置入 m$, 要求 n<20
     DIM  m$(20)
     CALL  ToArray(n, mn$, m$( ))
     ! 调用辅助子程序 frame 画框
     SET  COLOR  "yellow/magenta"
     CALL  frame(2, 1, 22, 78)
     ! 显示各个项目
     SET  COLOR  "yellow/blue"
     PRINT  tabe(1, 1); repeat$(" ", 80);
     LET  t=78/n
     FOR  i=1  to  n
          PRINT  tab(1, (i-1)*t+2);  m$(i)
     NEXT  i
     ! 循环选项
     LET  i=max(1, choice)        ! i 初始化, 可利用选取标志的传入值来规定 i 的初值
     DO
          ! 用醒目的红色背景突出显示第 i 项
          SET  BACK  "red"
          PRINT  tab(1, (i-1)*t+2);  m$(i)
          LET  i0=i
          ! 调用辅助子程序 pick 接收键盘按键, 得到 i 和选取标志 choice
          CALL  pick(n, i, choice)
          IF  choice>=0  then  EXIT  SUB
          ! 将原来突出显示的项恢复正常显示
          SET  BACK  "blue"
          PRINT  tab(1, (i0-1)*t+2);  m$(i0)
     LOOP
END  SUB
!
SUB  menu2(row, col, n, l, mn$, choice)
     ! 调用辅助子程序 ToArray, 将 mn$ 中的 n 个项目名称拆开并置入 m$, 要求 n<20
```

```
    DIM  m$(20)
    CALL  ToArray(n, mn$, m$( ))
    ! 调用辅助子程序 frame 画框
    SET  COLOR  "white/black"
    CALL  frame(row, col, n, l)
    ! 显示各个项目
    SET  COLOR  "yellow/blue"
    FOR  i=1  to  n
        PRINT tab(row+i, col+1); (m$(i)&repeat$(" ", l))[1:l]
    NEXT  i
    ! 循环选项
    LET  i=max(1, choice)        ! i 初始化，可利用选取标志的传人值来规定 i 的初值
    DO
        ! 用醒目的红色背景突出显示第 i 项
        SET  BACK  "red"
        PRINT  tab(row+i, col+1); (m$(i)&repeat$(" ", l))[1:l]
        LET  i0=i
        ! 调用辅助子程序 pick 接收键盘按键，得到 i 和选取标志 choice
        CALL  pick(n, i, choice)
        IF  choice>=0  then  EXIT  SUB
        ! 将原来突出显示的项恢复正常显示
        SET  BACK  "blue"
        PRINT  tab(row+i0, col+1); (m$(i0)&repeat$(" ", l))[1: l]
    LOOP
END  SUB
! 将字符串 mn$ 中的 n 个项目名称拆开，分别置入子符串数组 m$ 的对应元素中
SUB  ToArray(n, mn$, m$( ))
    LET  menu$ =trim$(mn$)
    FOR  i=1  to n-1
         LET  m$(i)=menu$[1: pos(menu$, " ")-1]
         LET  menu$ =trim$(menu$[pos(menu$, " ")+1: maxnum])
    NEXT  i
    LET  m$(n)=trim$(menu$)
END  SUB
! 以(row, col)作为左上角位置，在屏幕上显示 n+2 行 l+2 列的框(框内为 n 行 l 列) .
SUB  frame(row, col, n, l)
    ! 边框符号"┌, 一,┐ , |, └,┐"的 ASCII 码为 218, 196, 191, 179, 192, 217
    PRINT  tab(row, col); chr$(218); repeat$(chr$(196), l); chr$(191);
    FOR  i=1  to  n
         PRINT  tab(row+i, col); chr$(179); repeat$(chr$(176), l); chr$(179)
    NEXT  i
    PRINT  tab(row+n+1, col); chr$(192); repeat$(chr$(196), l); chr$(217);
```

```
END  SUB
! 从键盘上接收一个键，然后根据这个键确定 i 和 choice 的值
SUB  pick(n， i， choice)
     LET  choice=-1
     GET  KEY  a
     SELECT  CASE  a
     CASE  331， 328                 ! ←或↑键
        LET  i=i-1
     CASE  333， 336                 ! →或↓键
        LET  i=i+1
     CASE  13                        ! 回车键
        LET  choice=i
     CASE  27                        ! ESC 键
        LET  choice=0
     CASE  else
     END  SELECT
     LET  i=mod(i-1， n)+1           ! i 在 1 到 n 的范围内循环
END  SUB
----------------------------------------------------------------
Ok.SAVE  menu.lib
```

(2) 菜单子程序的调用 以下两个例子分别说明了菜单子程序 menu1 和 menu2 的调用方法。这两个例子通过菜单选择不同的频率进行发音，直至选“Quit”项退出。

```
LIBRARY  ″menu.lib″
SET  MODE  ″80″
SET  CURSOR  ″off″ ! 关闭光标
LET  mn$ =″400 600 800 1200 Quit″
DO
   CALL  menu1(5， mn$， choice)
   SELECT  CASE  choice
   CASE  1
      SOUND  400， 1
   CASE  2
      SOUND  600， 1
   CASE  3
      SOUND  800， 1
   CASE  4
      SOUND  1200， 1
   CASE  0， 5
      EXIT  DO
   END  SELECT
LOOP
SET  COLOR  ″yellow / red / blink″
PRINT  tab(12， 30); ″ Program  End !″
END
```

```
LIBRARY  ″menu.lib″
SET  MODE  ″80″
SET  CURSOR  ″off″
LET  mn$ =″400 600 800 1200 Quit″
DO
   CALL  menu2(4， 10， 5， 10， mn$， choice)
   SELECT  CASE  choice
   CASE  1
      SOUND  400， 1
   CASE  2
      SOUND  600， 1
   CASE  3
      SOUND  800， 1
   CASE  4
      SOUND  1200， 1
   CASE  0， 5
      EXIT  DO
   END SELECT
LOOP
SET  COLOR  ″yellow / red / blink″
PRINT  tab(12， 9); ″ Program  End !″
END
```

(3) *多级菜单驱动*　一个复杂的程序往往需要采用多级菜单驱动。下述程序是一个采用两级菜单驱动的绘图程序。第一级菜单有编辑数据、绘柱状图、绘曲线图、退出程序 4 项功能选择。当在第一级菜单下选择编辑数据功能时，屏幕将出现第二级菜单，包括显示数据、编辑数据、定义数据个数、退出二级菜单 4 项功能选择。

多级菜单驱动程序运行时，显示下一级菜单可能会覆盖原来的菜单，因而退出下一级菜单后需要将原来的菜单恢复。为此，本程序使用了能够在正文模式下存储和重显屏幕内容的子程序：

KeepBox(row, col, n, l, b$)——将左上角为(row, col) 的 n+2 行 l+2 列屏幕内容存入字符串变量 b$ 中；

ShowBox(b$)——将存入 b$ 中的屏幕内容重新原样显示出来。

这两个子程序建立在库 KeepBox.lib 中。

另外，本程序还需使用库 viewdata.lib(包含子程序 ShowData、EditData 和 InputData，见本章例 25)和库 drawdata.lib(包含本章例 7 的子程序 curve 和例 9 的子程序 bar)。

程序：

```
LIBRARY "menu.lib"
LIBRARY "keepbox.lib"
LIBRARY "viewdata.lib"
LIBRARY "drawdata.lib"
!
LET m$="Data Bar Curve Quit"        ! 一级菜单的项目
DIM v(1: 0)
SET MODE "80"
SET CURSOR "off"
DO
    CALL menu1(4, m$, c)            ! 显示一级菜单(横移式)并选取菜单项目
    SELECT CASE c
    CASE 0, 4                       ! 按ESC键或选择第4项Quit则退出
        EXIT DO
    CASE 1                          ! 选择第1项Data时调用sub1编辑数据
        CALL sub1(v( ))
    CASE 2                          ! 选择第2项Bar时调用bar绘制柱状图
        SET MODE "graghics"
        CALL bar(v( ))
        GET KEY a
        SET MODE "80"
    CASE 3                          ! 选择第3项Curve时调用curve绘制曲线图
        SET MODE "graphics"
        CALL curve(v( ))
        GET KEY a
        SET MODE "80"
    END  SELECT
LOOP
! 显示退出信息
CALL  frame(9, 26, 5, 30)
SET  COLOR "yellow/red/blink"
PRINT  tab(12, 36); " Good  Bye ! "
END
! 子程序 sub1 用于编辑数据
SUB  sub1(v( ))
    LET  n=size(v( ))
    LET  m$ ="Show  Edit  Define  Quit"! 二级菜单的项目
    CALL  KeepBox(2,2,4,12,b$)          ! 保存二级菜单的背景
    DO
        CALL  menu2(2,2,4,12,m$,c)  ! 定位显示并选取二级菜单(下拉式)
        IF  c=0  or  c=4  then  EXIT  DO! 按ESC键或选第4项Quit则退出
        CALL  KeepBox(4,30,max(n,2),21,b1$)! 保存数据表格的背景
        SELECT  CASE  c
        CASE  1                     ! 选第1项Show则显示数据表格
            CALL  ShowData(v( ),4,30,n)
            GET  KEY  w
        CASE  2                     ! 选第2项Edit则在表格上编辑数据
            CALL  EditData(v( ),4,30,n)
        CASE  3                     ! 选第3项Define则定义数据个数
            SET  COLOR  "yellow/black"
            DO
                CALL  frame(4,30,2,21)
                PRINT  tab(5,31); " How many data ? "
                PRINT  tab(6,31); " (1<=n<=20)"
                CALL  InputData(5,48,n,0)
            LOOP  until  n>0  and  n<=20
            MAT  redim  v(n)
        END  SELECT
        CALL  ShowBox(b1$)          ! 恢复数据表格的背景
    LOOP
    CALL  ShowBox(b$)               ! 恢复二级菜单的背景
END  SUB
-----------------------------------------------------
Ok.RUN
```

下面给出库 KeepBox.lib 的程序内容。因为其中子程序的设计涉及屏幕显示的某些细节(KeepBox 和 ShowBox 通过调用标准函数 peek 和标准子程序 poke 来直接对正文模式下屏幕的存储单元进行存取。正文模式下屏幕存储单元的地址为 753664～757663

且按顺序每两个单元表示一个屏幕字符)，因此可以不加深究直接运用。

```
EXTERNAL
SUB  KeepBox(row, col, n, l, b$)
     LET  b$ =chr$(row) & chr$(col) & chr$(n) & chr$(l)
     FOR  i=row  to  row+n+1
          FOR  j=col  to  col+l+1
               LET  adr=753663+(i-1)*160+2*j-1
               LET  b$ =b$ &chr$(peek(adr))&chr$(peek(adr+1))
          NEXT  j
     NEXT  i
END  SUB
!
SUB  ShowBox(b$)
     LET  row=ord(b$[1:1])
     LET  col=ord(b$[2:2])
     LET  n=ord(b$[3:3])
     LET  l=ord(b$[4:4])
     LET  t=4
     FOR  i=row  to  row+n+1
          FOR  j=col  to  col+l+1
          LET  t=t+2
          LET  adr=753663+(i-1)*160+2*j-1
          CALL  poke(adr, ord(b$[t-1:t-1]))
          CALL  poke(adr+1, ord(b$[t:t]))
          NEXT  j
     NEXT  i
END  SUB
----------------------------------------------
Ok.SAVE  KeepBox.lib
```

习 题 10

(1) 为使语句 BOX LINES −10，10，−10，10 在屏幕上显示出一个正方形，应怎样设置窗口语句？(设屏幕的纵横比是 2／3)。

(2) 用折线顺序连结(1，1)，(1−1)，(−1，−1)，(−1，1)这四点，构成一正方形，再以其四边的中点为新的顶点画一正方形，依此类推，直至画出五个正方形。

(3) 已知 n 个点的坐标(x_i，y_i)，(i＝1，2，…，n)，以每点到原点距离的长短为序(从小到大)，在屏幕上将这 n 个点用折线连接起来。要求画出框图，并编写程序。

(设置窗口语句为：SET WINDOW −1，1，−1，1；坐标数据由 DATA 语句给出；数据范围：$-1<x_i<1$，$-1<y_i<1$)。

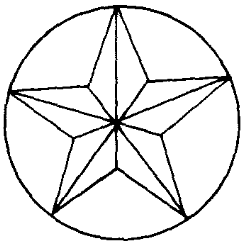
(4) 试编写程序，在屏幕上显示圆内五角星图案。

窗口设置：SET WINDOW -1.5，1.5，-1，1

背　　景：黑色

圆和中棱：黄色

边和内部：红色



(5) 设屏幕模式为“GRAPHICS”方式，窗口范围：0<x，y<20，以 (10，10)为圆心画一半径为 5 的圆，在窗口范围内由随机函数产生 100 个点，编程统计这些点落入圆内(包含圆上)的百分比。(要求用“RED”色画圆，分别用“YELLOW”和“GREEN”色画落入圆内和圆外的点)。

(6) 在屏幕上画一圆，将圆 n 等分(n 为素数)，从某一等分点出发，顺时针连接其后面的第 K 个等分点($k>1$)，如此继续下去，直到回到起点。

(7) 画一边长为 d 的正 n 边形，在其内作一个边长为 d/k 的内接正 n 边形，然后在新的多边形内继续画相应的内接多边形，共画 m 个正 n 边形。

第11章 文件

由于计算机内存容量有限，并且一旦主机断电，内存信息随即消失，因此，需要有另外的设备来长期保存信息。一般的计算机系统是采用磁盘、磁带等外部存贮设备来达到这一目的的。

True BASIC 系统将一个程序、一批数值数据或一组文字符号作为一个独立的有名字的信息集合存贮在磁盘上。这种存贮在磁盘上的有意义的信息集合称为计算机“文件”。True BASIC 系统支持对磁盘的操作，本章主要讨论磁盘文件。

11.1 概述

操作系统为了管理磁盘信息，当一批数据或一个程序存入磁盘时，总是以一个有名字的独立的信息集合形式存在，即文件。一张磁盘上通常可以存贮若干个文件，每个文件必须有唯一的文件名。文件命名应遵循操作系统规定的规则。从计算机系统的角度看，“文件”具有更广的含义，还可以把主机以外的一些设备(如屏幕／键盘，打印机)也看成是特殊的文件。利用高级语言提供的文件操作语句能够对文件进行读／写(或称为输入／输出)操作。

True BASIC 系统按文件的组织形式不同，将文件分为三类：正文文件、记录文件、字节文件。对它们进行读写操作一般都包括三个步骤：打开文件；读／写操作(输入／输出)；关闭文件。这些步骤的作用分别是：

(1)打开文件的目的是，确定程序将要操作的是磁盘上的哪一个文件，说明所打开的文件的组织形式和存取方式；同时，建立一个“通道”沟通程序与文件的联系，使程序通过这个通道对文件进行读／写操作。

(2)文件打开后，就可以对文件进行读／写操作。True BASIC 对组织形式不同的文件进行读写操作的语句是不同的，对于正文文件是用 INPUT 语句和 PRINT 语句完成输入／输出；对于记录文件和字节文件是用 READ 语句和 WRITE 语句完成输入／输出的。

(3)对一个文件读写操作完毕，需要用一个 CLOSE 语句关闭相应的通道，切断程序与文件的联系，以保护文件内容，避免对文件误操作。

11.2 文件的公共操作

11.2.1 打开文件

打开磁盘文件时，除确定文件的名称、组织形式和存取方式外，还需要确定一个“通道”，以沟通程序与文件的联系。通道是内存与磁盘之间信息传输的桥梁。建立通道后，在程序中对文件的输入／输出操作，就变成了对通道的输入／输出操作，关闭通道就是关闭对应的文件。

True BASIC 允许在一个程序中同时打开 10 个文件，同时打开的文件用不同的通道号加以区分。通道号是一个带“#”号标志的 0 至 1000 以内的整数。其中“#0”是特指屏幕／键盘的通道，这个通道总是打开的(不能由关闭语句关闭)。打开文件要用 OPEN 语句来实现。

语句一般格式：

OPEN ＃expr: NAME exprS [, ACCESS accmode$, CREATE cremode$, ORGANIZATION orgmode$, RECSIZE recsize]

其中，除 OPEN ＃expr:NAME expr$ 是必需的之外，后面方括号内的项均为可选项(在 OPEN 语句中可省略)。各项之间用逗号分隔，它们出现的次序是任意的，只要在 NAME expr$ 项之后即可，这些选择项每项的后缀又有几种选择(表 11-1)。

表 11-1 打开文件语句中后缀的意义

项 目	后 缀	意 义
ACCESS	INPUT	输入
(存取方式)	OUTPUT	输出
	OUTINN	输入／输出*
CREATE	NEW	建立新文件
(建立方式)	OLD	使用旧文件*
	NEWOLD	建立新文件或使用旧文件
ORGANIZATION	TEXT	正文文件
(组织方式)	RECORD	记录文件
	BYTE	字节文件

* 表示该项省略时，系统将自动选取的后缀。

NAME expr$ 中的串表达式 expr$，应当是一个符合操作系统规定要求的磁盘文件名，即程序要操作的文件．如果按文件方式使用外部设备，必须用操作系统给每一设备规定“文件名”(按操作系统规定：CON 为屏幕／键盘。LPT1 或 PRN 为第一台并行打印机。COM1 为第一异步通信接口等)。True BASIC 规定：对于一般的磁盘文件，打开文件时不写明扩展名，则隐含扩展名为“TRU”(用 SAVE 命令存贮已编译的程序时，扩展名隐含为“TRC”)。如果一个文件只有主名没有扩展名，则应在主名后加上一个“.”号。

CREATE cremode$ 项中的后缀 cremode$，用于说明所打开的文件是新文件还是旧文件．如果将要打开的文件是新的(磁盘上无此文件名)，应当用 NEW，以建立一个新文件；如果将要打开的文件在磁盘上已经存在，则用 OLD，以便使用它；如果事先并不知道磁盘上是否存在该文件，就使用 NEWOLD，那么将打开的文件既可以是旧文件也可以是新文件．如果磁盘上有同名的文件，则打开该文件；如果磁盘上无同名的文件，则建立并打开一个新文件．

ACCESS accmode$ 项中的后缀 accmode$，用于指明将要打开的文件是用作输入数据还是用作输出数据．这里的所谓输入／输出，是相对主机而言的。主机从外部获取数据的过程称为输入，相应的设备称为输入设备；相反，主机向外部送出数据的过程称为输出，相应的设备称为输出设备．

选用 ACCESS input，是对文件作输入。选用 ACCESS output，是对文件作输出。而选用 ACCESS outin，则既可对文件作输入，也可对文件作输出．

ORGANIZATION orgmode$ 项中的后缀 orgmode$，用于指明文件的组织形式。它有三种选择：如果文件为正文文件，则用 text；若为记录文件则用 record；若是字节文件就用 byte．如果缺省这一项，则文件的组织形式将由第一个对该文件进行输入或输出操作的语句

来决定，系统会根据操作语句的类型自动确定文件的组织形式。为了简便，允许把关键字ORGANIZATION 简写成 ORG。

RECSIZE recsize 项只用于记录文件和字节文件。后缀 resize 说明每次对文件进行操作的信息量，以字节为单位。

11.2.2 关闭文件

CLOSE 语句用于关闭一个通道，即关闭与此通道相连接的文件。对一个文件读写操作完毕，就需要将其关闭，切断程序与文件之间的联系。关闭文件不仅是为了保证文件操作过程的完整性，也是为了保证文件数据的完整。尤其当文件打开用作输出时，在文件操作过程中，由于某种原因可能尚有部分数据驻留在内存中还没有存入文件，这时只有关闭文件方能保证所有的数据存入文件。另外，一个文件按特定的方式打开后，通常只能进行对应的操作。如果想改变文件的操作方式，就必须先将文件关闭后再用其他方式打开。文件被关闭之后，原通道号又可重新用于另外的 OPEN 语句打开其他文件。CLOSE 语句的一般形式为:

CLOSE#expr

其中，expr 值为 1～1000 的数值表达式。

应当指出，在外部函数或外部子程序中也可以打开文件通道，但是这种通道的作用范围仅限于该函数或子程序中，即局部通道。当从外部函数或外部子程序返回调用程序时无论是否执行 CLOSE 语句，系统都将关闭局部通道。另外，通道号还可作为一个参数传递给子程序。而子程序中表示通道的形式参数，须写成#号后跟一个 1～1000 的整数(如，#999，#321 等等)。这些参数象子程序的其他类型的参数一样工作，当子程序被调用时，将换上实际通道号。

11.2.3 文件指针

与 READ／DATA 语句的读数指针相类似，文件操作过程中也有一个文件指针。它总是指向下一步将要操作的数据，并随着文件读写操作的过程而发生变化。文件打开时，指针总是指向文件的第一项数据，以后每读／写一次，指针就下移一项。这里的一项数据对于组织形式不同的文件，其含义是不同的。一项数据可以是一个数据(正文文件)，一个记录(记录文件)，也可以是一个字节(字节文件)。每个文件的末尾都有一个尾标志。当指针指到尾标志时，就不能再对文件进行读操作了，否则将导致错误。判断文件指针是否指向尾标志，可以用下面的逻辑函数:

MORE#expr　　或　　END#expr

当通道#expr 对应的文件尚未指向文件尾标志时，函数 MORE#expr 值为真；如果文件指针正指着文件尾标志，则 MORE#expr 值为假。函数 END#expr 的逻辑值与 MORE#expr 正好相反。

另外，True BASIC 还提供了 2 个对三类文件都适用的操作文件指针的语句:

SET#expr: POINTER BEGIN

作用: 将指针拨到文件的开头。

SET#expr: POINTER END

作用: 将指针拨到文件的末尾。

11.2.4 文件的删除

当文件的内容不再需要时，可以将其删除，True BASIC 提供了两种删除磁盘文件的方

法。

对于已打开的文件，可以用语句：

ERASE#expr

将与通道#expr 相连接的文件内容全部删除，只剩下一个文件名。经过 ERASE#expr 语句删除的文件，虽然其内容和组织形式都不复存在，但仍可对它进行打开或关闭操作。当再次使用它时，应当重新定义其组织形式。

对于未打开的文件，可用语句：

UNSAVE expr$

删除文件名为 expr$ 的文件。此语句不仅删除文件的内容，同时也删除文件名，但应注意：已经打开而尚未关闭的文件不能用这个语句来删除。

11.3 正文文件

11.3.1 正文文件的概念

存贮在磁盘上的程序或数据信息都是按某种形式编码存放的。最常用的编码是 ASCII 码(美国标准信息交换码)。正文文件由一行行的 ASCII 码字符构成。每一行的结束由一对特定的 ASCII 码字符(回车符和换行符)标记。由于 ASCII 码是一种国际通用信息交换码，因此许多高级语言都使用正文文件，以利于程序和数据的交换。在 True BASIC 中用 SAVE 命令存盘的数据和未经编译的程序即是正文文件。

正文文件是按字符顺序存放的。对正文文件的操作，只能是顺序读取或顺序写入。因此，正文文件又称为顺序文件。

11.3.2 打开正文文件

打开文件的语句随使用文件的方式不同而不同。例如，当文件打开用作输入时，打开文件语句可以是：

OPEN#expr: NAME expr$, ACCESS input, CREATE old, ORG text

按这种方式打开的文件，必须保证该文件在磁盘上已经存在。

而当文件打开是用作输出时，打开语句可以是：

OPEN#expr: NAME expr$, ACCESS output, CREATE new, ORG text

一个正文文件打开后，只能进行输入操作和输出操作的其中一种。即，要么作输入，要么作输出，不允许两者兼而有之，除非将文件关闭之后，再重新打开，变换存取方式。

如果在文件使用之前，还不能确定用作输入还是输出，或者不能确定欲打开的文件在磁盘上是否存在，最好使用下列方式打开文件：

OPEN#expr: NAME expr$, ACCESS outin, CREATE newold, ORG text

用这种方式打开文件，既可用作输入，也可用作输出。具体用作输入还是输出，将取决于第一个对文件进行操作的语句。如果第一个对文件进行操作的语句是输入语句，则此后对该文件只能作输入操作；如果第一个对文件进行操作的语句是输出语句，则此后就只能对该文件进行输出操作。按此方式打开文件，无论文件在磁盘上存在与否，总能完成打开文件的操作。如果文件存在，则使它与程序建立联系；如果文件不存在，就设立一个新文件，同时以通道建立它与程序的联系。

11.3.3 输出／输入

(1)输出语句格式

PRINT#expr: expr1，expr2，…，exprN

它的功能与不带通道号#expr: 的 PRINT 语句类似，是将 expr1，expr2，…exprN 的值经过通道 expr 送到对应的文件中去，而不带通道号的 PRINT 语句则是将各表达式的值送往屏幕，两者的输出格式相同。PRINT 语句在屏幕上显示字符，PRINT#expr 则向文件送字符；PRINT 语句在屏幕产生空格，PRINT#expr 则向文件送空格；PRINT 语句在屏幕上造成回车换行的效果，PRINT#expr 则向文件送一对回车换行符。因此可以说，正文文件是屏幕的映象。通过下面的例子，可以更清楚地看出这一点。

```
PRINT "This is a example !"
PRINT "PI="; pi, "e="; exp(1)
PRINT 1; 2; 3; 4
END
```

上面的程序运行后将在屏幕上产生如下效果:

```
This is a example !
PI= 3.14159          e= 2.71828
1  2  3  4
```

而下面的程序是将结果存入文件 DEMO.TRU 中。

```
OPEN#1: name "DEMO.TRU", access outin, create newold
PRINT#1: "This is a example !"
PRINT#1: "PI="; pi, "e="; exp(1)
PRINT#1: 1; 2; 3; 4
CLOSE#1
END
```

程序运行结束后，在命令窗口使用 OLD DEMO.TRU 命令，将所产生的 DEMO.TRU 文件的内容显示在编辑窗口上，便会看到与上一程序的显示结果完全相同。这就是所谓映象的含义。

例 1 建立正文文件 POSITIVE.TRU，存入下面 10 个数据中大于 0 的数。

−1，2，−3，−7，4，5，−9，6，−3，1

程序:

```
OPEN#1: name "POSITIVE.TRU", access output, create new, org text
FOR i=1  to 10
    READ number
    IF number>0 then PRINT #1: number
NEXT i
CLOSE#1
DATA -1, 2, -3, -7, 4, 5, -9, 6, -3, 1
END
```

程序运行后，POSITIVE.TRU 文件中的数据为:

2(回车换行符)

4(回车换行符)

5(回车换行符)

6(回车换行符)

1(回车换行符)

上例中，数据文件 POSITIVE.TRU 是程序运行后产生的新文件。在使用盘上的旧文件时，为了使输出数据之前文件是空的，应当使用 ERASE#expr 语句先将文件内容删除。如果需要在已有数据之后追加新的数据，则打开文件后，应先使用 SET# expr: POINTER END 语句将文件指针拨至文件末尾，然后再向文件输出新数据。

例 2　编一程序，对从键盘上输入的一个句子进行处理：将其中的空格滤掉，把各单词分离出来，再把它们追加到单词库 LIB.TRU 中去。

程序：

```
OPEN#2: name "LIB.TRU", access outin, create newold
SET#2: POINTER end
INPUT aS                                  ! 输入一个句子
LET e=len(a$)
LET s$=""
FOR i=1 to e
    LET p$=aS[i:i]                        ! 取一个字符
    IF p$<>" " then                       ! 是否为空格
       LET s$=s$&p$                       ! 拼接单词
    ELSE
       PRINT#2: s$                        ! 将单词存入 LIB.TRU
       LET s$=""                          ! 将s$清为空
    END IF
NEXT i
PRINT#2: s$                               ! 将最后一个单词存入 LIB.TRU
CLOSE#2                                   ! 关闭文件
END
```

对于绝大多数的屏幕输出语句，在关键字 PRINT 后加上通道号就可以变成对正文文件输出的语句。如：

MAT PRINT A; 和 PRINT USING "###.##": X;

语句加上通道号#expr，则将屏幕的显示映象存贮到正文文件中。

MAT PRINT#expr: A; 和 PRINT#expr, USING "###. ##": X;

(2)*输入语句格式*

INPUT#expr: var1, var2, …, varN

它的功能和键盘输入语句 INPUT 相似，区别仅在于文件输入语句 INPUT#expr 是从文件的当前指针起顺序地读取数据，并依次地赋给相应的变量；而 INPUT 则是从键盘上获取数据。由于键盘也可以看作是一个只能作输入的“文件”，因此 INPUT 语句可以认为是 INPUT#expr 的一种特殊情况。当打开的文件是 CON（屏幕／键盘）时，它的作用即与 INPUT 相同，只是屏幕上不出现 INPUT 语句的输入提示符“ ? ”而已。

由第5章知道，从键盘上为 INPUT 语句后的变量输入数据时，要求数据之间隔以逗号，键入最后一个数据之后应按回车键，这里对 INPUT#expr 也有同样的要求。例如：利用下面两个输入语句为两组变量 A，B，C 和 D，E 提供数据：

```
INPUT#1: A, B, C
INPUT#1: D, E
```

则文件内部存贮形式是：

数据1，数据2，数据3　回车换行符　数据4，数据5　回车换行符

而生成这个数据文件的程序，可以写成：

```
OPEN#1: name "DEMO.TRU", access outin, create newold
ERASE#1
READ v1, v2, v3, v4, v5
PRINT#1: v1; ","; v2; ","; v3
PRINT#1: v4; ","; v5
CLOSE#1
DATA 1, 2, 3, 4, 5
END
```

此外，也可以用直接在屏幕的编辑窗口键入数据行的方法，建立数据文件：

```
■  1,2,3(按回车键)
■  4,5(按回车键)
■
```

用 SAVE 命令，将编辑窗口的内容存入数据文件中。

对于工程计算方面的大批量原始数据，使用这种方法建立数据正文文件尤其合适。

例3　将例1文件 POSITIVE.TRU 中的偶数挑出来，并输出到打印机。

程序：

```
OPEN#1: name "LPT1", access output
OPEN#2: name "POSITIVE.TRU", access input, create old
DO while more#2
    INPUT#2: x
    IF mod(x,2)=0 then PRINT#1: x,
LOOP
PRINT#1
CLOSE#1
CLOSE#2
END
```

正文文件除了用于存放数值型的数据外，还可用于存放字符型数据，例如：一个单词库，一篇文章等。当需要对文件的字符按行进行处理时，可以使用行输入语句：

```
LINE INPUT#expr: var1$, var1$, ..., varN$
```

它的功能是顺序将文件的数据以行为单位赋给 LINE INPUT#expr 语句后的诸变量，在文件里回车换行符是行结束的标志。当读取数据遇到回车换行符时，即认为已读完一行。

应该指出，用 MAT PRINT#expr:array 语句存入文件中的数据，是不能直接用

MAT INPUT#expr: array 语句读进来的，因为用 MAT PRINT#expr 将数组数据存入文件时，每行的数据之间是用空格隔开的，而 MAT INPUT#expr 将数据读入数组，则要求除了最后一个数据之外，每个数据之后应有一个逗号，各行(除了最后一行)的最后一个数据也不例外。为此，要想使 MAT PRINT#expr 存入的数据能为 MAT INPUT#expr 直接使用，就必须预先把文件的内容作处理，使得每个数据之后加上逗号。例 4 就是描述这一处理过程的程序。

例 4 对用矩阵语句存入文件的数据作预处理，然后提供给矩阵语句作输入用。

程序:

```
OPEN#2: name "OLD", access input, create old
OPEN#3: name "NEW", access output, create newold
ERASE#3
LET true=0
DO while more#2
     LINE INPUT#2: l$
     LET c=len(l$)
     LET p=1
     IF true=1 and c>0 then
         PRINT#3: ","
     END IF
     DO while c>0
         LET l$=trim$(l$[p:c])
         LET p=pos(l$, " ",1)
         IF p>0 then
             LET n$=l$[1:p-1]
             PRINT#3: n$; ",";
         ELSE
             LET n$=l$
             PRINT#3: n$;
         END IF
     LOOP until p=0
     LET true=1
LOOP
CLOSE#2
CLOSE#3
END
```

这里假设在程序运行之前，已利用 MAT PRINT#expr 语句将一个数组的数据存入文件 OLD.TRU 中，经过处理后的数据放在 NEW.TRU 文件中。由于在最后一个数据之后不能加逗号，因此上述程序采用的处理办法是：利用 POS 函数寻找空格，将数据一个一个截取出来，存入文件 NEW.TRU 之中，并随后存一个“,”号，而每行最后一个数据之后的逗号则放在处理下一行之前根据情况再加。这是由语句 PRINT#3:“,”实现的，但在处理第一行之前和处理最后一行之后，不能执行这个语句，可通过逻辑判断 TRUE=1 AND c>0 来实

现。在处理第一行之前，TRUE=0，其后均为1。条件c>0是判断是否处理过最后一行的标志。因为在最后一行处理完之前，读入的行长度是大于零的。当最后一行数据处理之后，文件指针指着最后一行的回车换行符，此时，MORE#2逻辑值尚为真，再读一行时，行长度即为零(在正文文件中回车换行符只作为行结束标志，而不作为数据字符)。因此，根据c是否大于零就可以判断程序是否已处理完最后一行。从这个例子可以看出，文件数值数据既可以读出来赋给数值型变量，也可以作为字符串行读出赋给字符串变量。

例5 假设磁盘上有两个有序的数据文件，文件中的数据按照一行一个数据的形式顺序存放。要求对它们作归并处理，产生一个包含前两个文件所有数据的新的有序文件。归并的过程要求：不能将两个有序文件的所有数据调入内存作排序处理，而是在外存中进行排序(因此又称为外排序)。

程序：

```
OPEN#1:name "af", access input
OPEN#2:name "bf", access input
OPEN#3:name "cf", access outin, create newold
SET#1:POINTER begin
SET#2:POINTER begin
ERASE#3
INPUT#1: a
INPUT#2: b
DO
    IF a<=b then                        ! 如果a小于等于b
        PRINT#3: a                      ! 存a的值
        IF end#1 then                   ! 如果第一个文件指针指向文件尾
            PRINT#3: b                  ! 则将第二个文件的剩余数据
            DO until end#2              ! 全部存入第三个文件中
                INPUT#2: b
                PRINT#3: b
            LOOP
            EXIT DO
        ELSE                            ! 否则(第一个文件指针未指向文件尾)
            INPUT#1: a                  ! 读下一个数据
        END IF
    ELSE                                ! 如果a大于b
        PRINT#3: b                      ! 存b的值
        IF end#2  then                  ! 如果第二个文件指针指向文件尾
            PRINT#3: a                  ! 则将第一个文件的剩余数据
            DO until end#1              ! 全部存入第三个文件中
                INPUT#1: a
                PRINT#3: a
            LOOP
            EXIT DO
        ELSE                                 ! 否则(第二个文件指针未指向文件尾)
                INPUT#2: b                   ! 读出下一个数据
        END IF
    END IF
LOOP                                         ! 进行下一次比较
CLOSE#1
CLOSE#2
CLOSE#3
END
```

(3) 关于键盘／屏幕和打印机的说明

键盘／屏幕，打印机这两种外部设备也可以作为两个特殊的文件。下面简述对它们的打开方式。

键盘／屏幕的文件名规定为 CON，只能打开作为输入。打开键盘／屏幕的语句格式:

OPEN#expr: NAME “CON”, ACCESS input

打印机的文件名规定为 PRN，只能打开作为输出。打开打印机语句格式:

OPEN#expr: NAME “PRN”, ACCESS output

* 11.4 记录文件

记录文件是记录的有序集合。记录是对记录文件进行操作处理的基本单位。在记录文件中，每个记录都有一个编号，称为记录号。在同一文件中的各个记录具有相同的数据结构，它反映了事物的某些属性。例如：把一张学生成绩表存入记录文件(表 11-2)。

表 11-2 学生成绩表

记录号	学号	姓名	分数	指针位置
1	8801	李　刚	87	←-----BEGIN
2	8802	方梅花	78	
3	8803	张超灵	56	←-----SAME(当前位置)
4	8804	伍正果	90	←-----NEXT
5	8805	李　莉	68	
(文件尾标志)				←-----END

一个记录可以有多个数据项。表中每一行是一个记录，每个记录均有学号、姓名、成绩 3 个数据项。记录的数据类型可以是数字型，也可以是字符型。

打开记录文件的一般格式:

OPEN#expr: NAME expr$, CREATE newold, ORG record, RECSIZE expr1

记录文件中的每个记录都不能超过某一规定的长度(记录长度)。记录长度可在建立文件时由 OPEN 语句的 RECSIZE expr1 项的后缀 expr1 指明，也可用下述语句来规定:

SET#expr : RECSIZE n

其中，n 规定一个记录允许的最大长度，单位是字节。当 n 为非整数时，系统自动四舍五入取整。如果文件已经存在，记录长度已建立在文件中，就不允许再用 SET#expr:RECSIZE n 语句来重新说明记录长度。如果文件内容被删空，其记录长度方可重新设定。记录文件一打开，文件指针即指向第一个记录，以后每输入或输出一个记录，指针就下移一个记录。

记录文件的输出语句格式：

WRITE #expr：expr1，expr2，…，exprN

其作用是：将 N 项数据写入从文件指针当前所指位置起的 N 个记录中去。字符串和数字可以混合写入文件，每个数值数据占 8 个字节，字符串的每个字符占一个字节(字符串两端的双引号不存入)。记录文件允许在已经有内容的记录中重新写入新数据覆盖原内容。

记录文件的输入语句格式：

READ #expr：var1，var2，…，varN

其作用是从 expr 号记录文件当前指针所指记录起，顺次读取 N 个记录的数据，并将这些数据按序赋给诸变量。输入数据的过程要求变量的类型应与对应的记录数据类型相吻合。即，数字型数据只能赋给数字型变量，字符串型数据只能赋给字符串型变量(因此，语句格式中的 VarI 既可是数值型变量，也可以是字符串变量，这时变量后应带符号 $)。

对记录文件的文件指针操作语句除了 11.2 节中介绍的 2 种外，还有 3 个专门用于操作记录文件指针的语句：

SET #expr：POINTER same

作用：指向上一记录(即刚才操作过的记录)。

SET #expr：POINTER next

作用：指向下一记录。

SET #expr：RECORD n

作用：指向第 n 号记录(第一个记录编号为 1)。

例 6　将某商店商品的名称和数量登记在记录文件上，以便重复使用。

程序：

```
OPEN#2: name "stuff", organization record, create newold
ERASE#2
SET #2:RECSIZE 20
DO while more data
   READ item$, quantity
   WRITE #2:item$, quantity
LOOP
CLOSE#2
DATA basktballs, 20     ! 商品名，数量
DATA footballs, 40
END
```

程序中，语句 WRITE#2 一次写入 2 个记录，前一个记录为字符型，后一个记录为数值型。记录的长度都是 20 个字节。程序运行形成一个商品名称和数量相间出现的记录文件(文件名为：stuff.tru)。

例 7　读取并显示例 6 程序运行产生的记录文件(stuff.tru)中的数据。

程序：

```
! read file written by previous program.
OPEN #1:  name "stuff", org record,  create newold
DO while more#1
```

```
    READ#1: item$, quantity
    PRINT item$, quantity
LOOP
CLOSE#1
END
```

程序中，语句 READ#1 一次读出 2 个记录，由于奇数号记录为字符串，所以只能赋给字符串型变量；而偶数号记录为数值型数据，因此只能赋给数值型变量。

例 8 建立一个反映职工基本情况的文件（文件包括 4 个数据项：姓名，性别，年龄，工资），然后读出并显示其中某些数据。

程序：

```
! 定义一个函数将四项数据处理成一个综合字符串
DEF pack$(name$, sex$, age, salary)
    LET name$ = using$("<#########", name$)
    LET sex$ = using$("<##", sex$)
    LET age$ = num$(age)
    LET salary$ = num$(salary)
    LET pack$ = name$&sex$&age$&salary$
END DEF
! 定义一个子程序，将一个综合字符串按字节分段拆开，还原为四项数据
SUB unpack(s$, name$, sex$, age, salary)
    LET name$ =s$[1:10]
    LET sex$ =s$[11:13]
    LET age=num(s$[14:21])
    LET salary=num(s$[22:29])
END SUB
OPEN #1: name "rsb", create newold, organization record
ERASE #1
SET #1: RECSIZE 29
DO while more data
    READ name$, sex$, age, sal
    WRITE #1: pack$(name$, sex$, age, sal)
LOOP
SET #1: POINTER begin
DO while more #1
    READ#1: s$
    CALL unpack(s$, name$, sex$, age, sal)
    PRINT name$, sal
LOOP
CLOSE#1
DATA "Zhangsan", "nan", 25, 80
DATA "Lisi", "nu", 40, 120
DATA "Wangwu", "nan", 50, 120
END
```

*.**例 9** 设记录文件 DOCUMENT.TRU 里已按学号顺序存入一个班某门课的成绩档案。从键盘输入任意一个学号，试用二分法查找该学号所对应的学生该门课的成绩。

程序:

```
OPEN#1: name "DOCUMENT", access outin, create newold, org record
INPUT prompt "Student number =": sn
ASK#1: FILESIZE n                         !询问文件中记录的个数
SET#1: POINTER begin
LET find=0
CALL unpack(number, name$, score)
IF sn>=number then                        !SN大于第一个记录的学号吗
   SET#1: POINTER end
   SET#1: POINTER same
   CALL unpack(number, name$, score)
   IF sn<=number then                     !SN小于等于最后一个记录的学号吗?
      LET p1=1                            !首指针
      LET p2=n                            !尾指针
      DO                                  !二分查找
         LET p=int((p1+p2)/2)
         SET#1: RECORD p
         CALL unpack(number, name$, score)
         IF sn<=number then LET p2=p else LET p1=p+1
      LOOP until p1=p2
      SET#1: RECORD p1
      CALL unpack(number, name$, score)
      IF sn=number then                   !找到的记录满足条件吗?
         PRINT name$, score               !显示姓名和成绩
         LET find=1                       !设置"找到"标志
      END IF
   END IF
END IF
IF find=0 then                            !没有找到则显示"错误"信息
   PRINT "Can't find number"; sn
END IF
CLOSE#1
SUB unpack(number, name$, score)          !记录数据"开包"子程序
    READ#1:s$
    LET number=num(s$[1:8])
    LET name$ =s$[9:16]
    LET score=num(s$[17:24])
```

```
END SUB
END
```

在程序中，打开文件之后，首先输入欲查询的学号，然后对文件的第一个记录和最后一个记录作判断：所查询的学号是否在档案学号的范围之内。如果是，则以 P1 为首指针，P2 为尾指针，P 为中指针进行二分查找。当找到了该学号，即显示结果，否则显示“错误”信息。程序中查询最后一个记录是用 SET#1:POINTER end 和 SET#1:POINTER same 两个语句实现的。当已知记录个数时，也可以用 SET#1:RECORD n达到同样的目的。

*11.5 字节文件

从机器内部存贮的角度来看，信息的存放，无论是对哪一类文件，都是以字节为单位将数据存贮于文件中。True BASIC 提供了字节文件和对字节文件的操作，把磁盘文件视为一个由字节构成的序列(字节流)，对文件的读／写操作均以字节为单位进行。

字节文件象其它文件一样，可以打开，关闭和删除，也可以操作文件指针。但是 True BASIC 系统规定：对字节文件的读写操作，总是以字符串的形式进行的。

打开字节文件的一般格式：

OPEN#expr: NAME expr$， ACCESS outin， CREATE newold， ORG byte

字节文件的输出语句格式：

WRITE #expr : exprl$，expr2$， ……

其作用是将字符串表达式的结果值(字符串)按序写入字节文件中。

字节文件在输入数据之前，需要事先规定每次读多少个字节的内容，赋给一个字符串变量。可以用 SET 语句规定。

语句格式：

SET#expr : RECSIZE exprl

其中，#expr 为文件号，exprl 为字节数。

字节文件的输入语句格式：

READ#expr : var1$，var2$， …，varN$

其作用是从#expr 号文件每次读取固定字节数的文件内容，分别赋给字符串变量var1S，…，varN$。

如果需要临时改变读入字符串变量的字节数，可以使用如下格式的读入语句：

READ#expr， BYTES expr : var1$，var2$， …，varN$

如果是对未知长度的文件进行字节文件操作，可能会在程序运行过程中因为指针已经指到文件末尾(已无数据可读)而导致程序运行出错。为了避免这种情况发生，应当在程序中加入出错处理以屏蔽此种故障。

在字节文件里操作文件指针所用的语句，与记录文件中所用的一样，其含义如表 11-3 所示。

***例 10**　拷贝字节文件 from 中的内容到文件 to 中去。

程序：

```
! copy over a file using byte files
```

```
OPEN#1: name "from", access input, organization byte
OPEN#2: name "to", access output, create newold, organization byte
ERASE#2
DO
    WHEN error in
        READ#1, bytes 2048:buffer$
    USE
        LFT endflag=1
    END WHEN
    WRITE #2: buffer$
LOOP until endflag=1
CLOSE#1
CLOSE#2
END
```

其中，WHEN error in／USE／END WHEN 是出错处理语句。程序反复地从#1文件读出2K字节赋给字符串变量 buffer S，然后用 WRITE 语句将它写入#2文件，直至文件尾。这时再读#1文件即出错，程序因此转入出错处理：执行 USE 至 END WHEN 部分，将 endflag 置 1，在 loop 处检验 endflag 时，条件为真，退出循环，结束程序。

表 11-3　字节文件操作语句

语　句	意　义
SET#expr: POINTER SAME	指向上一字节(即刚操作过的字节)
SET#expr : POINTER NEXT	指向下一字节
SET#expr : RECORD Rn	指向第 Rn 个字节

11.6　打印机的使用

在 True BASIC 系统中，打印机是当作一个正文文件来使用的。打开打印机有以下几种方式：

OPEN#expr: NAME "PRN", ACCESS output

或

OPEN#expr: NAME "LPT1", ACCESS output

因为打印机是特殊的文件，所以 True BASIC 还特别提供另一种打开它的方法：

OPEN#expr: PRINTER

对打印机进行输出操作和对正文文件进行输出操作相同，所用的输出语句也是 PRINT#expr，它将 PRINT 语句在屏幕上的显示结果在打印机上反映出来。所有用于正文文件的输出语句都可用于打印机的输出。

利用下面的语句可以改变行宽：

SET#expr: MARGIN expr1

其中，expr1 的极限值随打印机的型号不同而不同。对普通宽行的西文打印机，expr1 最宽为 136 个字符，一行上的标准格式位置的个数可达 8。因此，改变行宽以后，同一程

序的屏幕显示效果就可能与打印机打印效果有区别。

另外，与屏幕显示类似，可以用如下格式语句改变标准格式位置之间的距离：

SET#expr: ZONEWIDTH expr1

当不作说明时为16个字符宽，说明后的宽度为expr1个字符。

11.7 高级编辑

True BASIC 除了具备大多数编辑软件都具有的编辑功能外，还扩充了许多编辑正文文件的命令，使得它的编辑功能更具有特色。本节简要介绍这些编辑命令。

11.7.1 10个功能键(F1～F10)

F1 键(Editing Window) ——将光标移到编辑窗口；

F2 键(History Window) ——将光标移到背景(命令)窗口；

F3 键(FIND) ——查寻程序中的字或字符串；

F4 键(MARK) ——标记程序的行模块；

F5 键(COPY) ——将被标记的行模块复制到光标所在行的下面；

F6 键(MOVE) ——将被标记的行模块移动到光标所在行的下面；

F7 键(RESTORE) ——恢复被 DEL (或 DELETE) 命令删除的行或行模块；

F8 键 ——设置断点；

F9 键(RUN) ——运行编辑窗口的程序；

F10 键(HELP) ——帮助了解 True BASIC 系统；

SHIFT+F1 键 ——向上寻找与光标所在行左端对齐或后缩较少的最近行；

SHIFT+F2 键 ——向下寻找与光标所在行左端对齐或后缩较少的最近行；

SHIFT+F3 键 ——返回书签（显示在行标志上的小圆圈）所在行；

SHIFT+F4 键 ——在光标所在行上建立书签。

11.7.2 14个主要高级编辑命令

(1) FIND 命令(F3 键)

在编辑窗口上搜寻指定的字或字符串，搜寻方式分两种：

①搜索某一个完整的字或数字　要搜寻的字不区分字母的大小写，因而寻找某字时可以用大写字母也可以用小写字母键入。

②搜寻某个字的一部分　搜寻某个字的一部分时，需用双引号括住被查寻的字符。此时，双引号内的符号串与程序中所要查寻的对象，在字母大、小写完全匹配才能被搜索到。

注意：True BASIC 总是从光标所在处开始搜索，直到程序的结束。因此，在搜索前按 home 键，将光标移到程序开始处，才能实行对整个程序的搜索。

(2) CHANGE 命令

CHANGE 作用于整个程序，其功能是修改一个字或一短语。即将当前程序中所出现的某一字或数字替换成新的字或数字。例如，将当前程序中所有的“print”改为“print#1:”。

```
Ok. CHANGE print , print#1:
```

这里 print 的大小写不区分。

同 FIND 命令一样， CHANGE 命令将不改变某字或数字的一部分，除非键入命令时使用双引号括住某些字符才可以进行字或数字的部分搜寻。

(3)TRY 命令

TRY 命令作用于整个程序，其功能亦是修改一个字或一短语。

TRY 命令的格式同 CHANGE 命令格式一样，不同之处在于，它每改一处之前都询问是否修改，回答 Y(Yes)表示改，回答 N(No) 表示不改；回答 Q(Quit)表示不改并退出 TRY 命令状态。

(4)标记行模块

标记一个由若干行组成的模块有 2 种操作方式：

①使用 F4(MARK)标记键　将光标移到要标记的行模块的首行，按下 F4 键，这时，该行行首即出现长方形的标志；再将光标移到要标记的行模块尾行，按一下 F4 键，此时首尾行之间的所有行均被加上长方形标志；然后，即可进行行模块的有关操作。

②使用 MARK 命令　当需要对某行模块(如函数、子程序或图画等)进行编辑操作时，可以采用 MARK 命令。例如，标记以 Subn1 为名的子程序，可在 Ok.右边键入：MARK Subn1，则整个 Subn1 子程序的行均被标记。

对作过标记的模块，可作移动、复制、删除等编辑处理。

(5)删除行模块

首先按(4)中的步骤标记好行模块，然后进行模块的删除操作。删除行模块有两种方式：

①将光标移到被标记的行模块的任一行的行标志符上，然后按 Del 键，此行模块即被删除。

②将光标移到命令窗口，键入 DELETE 命令。则所标记的行模块被删除。

(6) 复制行模块

F5 键是复制(COPY)键，如果在编辑窗口使用 F5，其功能为将已标记的行模块复制到光标所在行的下面。具体操作分两步：

①用 F4 键标记好要复制的行模块。

②将光标移到需要复制该模块的位置，按一下 F5 键。

这时光标所在行的下面即拷贝了一个所标记的行模块。

如果是在命令窗口使用 F5，其作用是在命令窗口的最后一行重现最后一次键入的命令字符。

(7)上／下移动行模块

F6 (MOVE)是上／下移动行模块的功能键。其作用是将被标记的行模块移动到光标所在行的下面。操作步骤与复制行模块相同。

(8)左右移动行模块

被标记的行模块可以整个向左或向右移动。具体步骤如下：

①将要移动的行模块作上标记；

②将光标移到该行模块的任一行的行标志上；

③向右移动，按“>”(大于)键，每按一次，整个行模块向右移动一格；

④向左移动，按“<”(小于)键，每按一次，整个行模块向左移动一格。

(9)编辑行模块

对于一个很长的程序，如果希望编辑其中的某一程序段，可以采用编辑行模块命令 EDIT。具体步骤:

①标记将要编辑的行模块。

②按 F2 (History window)键，将光标移到命令窗口，在 Ok.之后键入 EDIT 命令(显示: Now editing marked region)。

True BASIC 先将除了被标记行模块以外的其它内容暂时隐去，然后对被标记模块内容进行编辑，这时的所有编辑命令都仅对该模块起作用。

③编辑完该行模块后，按 F4 消除模块标记。

④按 F2 将光标移到命令窗口，再键入 EDIT 命令。命令窗口显示“ Now editing whole file ”，这时在编辑窗口会再现包括被隐去部分在内的整个程序。

(10)连接行

如果希望将 True BASIC 编辑窗口中相邻的两个语句行合并成一个语句行。可以将光标移到第二个语句行的行标志符上，键入回格键(←)即可将该行移至上一行的末尾。

(11)程序的规格化

True BASIC 系统提供了一个 FORMAT.TRU 程序，其功能是将用户程序重新规格化。具体操作如下:

①将用户程序调入编辑窗口;

②按 F2 键转入命令窗口，在最后一行的 Ok. 右边键入 DO FORMAT。

DO FORMAT 将把程序中的关键字改为大写形式，并按向右压缩的结构格式书写程序，整理注释行，使程序格式标准化。注意，在作规格化之前，应确认当前磁盘目录里有 FORMAT.TRU 程序。

(12)INCLUDE 命令

在当前程序中插入一个磁盘文件。插入的文件被放在光标所在行的下面。

(13)KEEP 命令

保留被编辑文件中已标记的行模块，删除其余的内容。它有 2 种操作方式:

①对于子程序、函数和图画等有名字的模块，可键入命令和模块名。

Ok. KEEP　子程序名函数名或图画名

②对于要保留的行模块，先作行模块标记，然后将光标移入命令窗口，在 OK.之后键入命令 KEEP。

(14)LOCATE 命令

寻找和显示包含某个字或短语的所有程序行。用来显示某个变量或关键字所在的行，便于编辑和调试程序。

习题 11

(1) 设文件 DEMO.DAT 中所存数据为 1，2，3，4，写出下列程序的运行结果。

```
OPEN#1:name "DEMO.DAT", access outin, create newold
FOR i=1 to 4 step 2
    INPUT#1: x
    PRINT x;
```

```
NEXT i
CLOSE#1
END
```

(2) 先将 *n*+1 个平方根存到 ABC.DAT，再从该文件取出数据显示在屏幕上，试选择适当的答案填空，使程序完整。该程序运行后，屏幕显示的数据个数为(　　)。

```
OPEN#1:name "ABC.DAT", access outin, create newold
LET n=100
PRINT#1:sqr(n)
FOR i=1 to n
     LET a=sqr(i)
     (            )
NEXT i
CLOSE#1
!
OPEN#2: name "ABC.DAT", access outin, create newold
INPUT#2: n
PRINT n
FOR i=1 to n
     (            )
     PRINT a
NEXT i
CLOSE#2
END
```

A) INPUT a
B) INPUT#1: a
C) INPUT#2: a
D) INPUT#1: SQR(a)
E) PRINT a
F) PRINT#1: a
G) PRINT#2: a
H) PRINT#2: a=SQR(a)
I) 11 个
J) 1 个
K) 100 个
L) 101 个

(3) 写一个程序，使它产生一个正文文件，其内容为一个给定文件的所有奇数行。

(4) 写一个程序，在某正文文件的所有行中寻找第 4 列，第 6 列和第 10 列分别为 A，B，C 的行，并将这些字符分别改为 X，Y，Z。

(5) 编写一个文件复制程序。即将输入的内容和格式保持不变，原样输出。

(6) 写一程序，将输入的两个文件首尾相接，作为一个新文件输出。

(7) 截取输入文件中以字符串 S$ 起始，以 E$ 终止的内容（包含 S$ 和 E$），若文件中不存在这样的部分，显示“Not found”，否则作为新文件输出。

(8) 设正文文件 DEMO.TXT 中每行的长度小于 60，试写一个打印 DEMO.TXT 的程序，使 DEMO.TXT 可分成两排印在每页打印纸上。

(9) 按照习题 7 中第 16 题的规则对某正文文件加密并产生一个新文件。

附录1　出错处理

一个程序在编制、调试、运行过程中，经常发生语法错误、逻辑错误和运行错误，必须及时予以纠正。程序运行时发生的错误(不包括因算法设计不当引起的运算结果错误)统称为运行错误。大多数运行错误都可以依靠设置一些条件判断加以预防。例如，在做除法运算时若分母为零，将导致“溢出”错误。但如果在做除法运算以前加一个处理分母为零的判断语句，就可以避免这一错误。然而，有的运行错误用这种办法是无法解决的，或者虽能解决但由于附设的语句太多而使程序显得累赘。下面介绍在算法逻辑正确的情况下，使用True BASIC的出错处理对程序的某一部分附加保护措施，以保证程序的正确运行的方法技术。

1.1　出错处理的一般结构

```
WHEN ERROR IN
      ......)
      ......}  (被保护模块)
      ......)
USE
     ......)
     ......}  (错误处理模块)
     ......)
END WHEN
```

在WHEN语句与USE语句之间的一组语句称为“被保护模块”，而USE语句与END WHEN语句之间的另一组语句称为“错误处理模块”。程序正常运行时，仅执行被保护模块里的语句而跳过错误处理模块，而一旦在被保护模块里发生运行错误，True BASIC就会立即转向相应的错误处理模块，去执行其中的语句进行出错处理，然后再执行END WHEN下面的程序。

在被保护模块里除了不能定义函数、子程序、图画外，任何语句都可以出现。WHEN语句在被保护模块中还允许嵌套使用。对错误处理模块的使用限制就更少了，如果用户需要，可以在其中定义函数、子程序或图画，但是错误处理模块里不允许再包含其他被保护模块。如果在被保护模块里有调用自定义函数或子程序，则相应的函数或子程序也被保护。

1.2　几个与出错处理有关的函数和语句

1.2.1　EXTYPE，EXTEXT$和EXLINE$函数

EXTYPE函数的功能：返回一个最近发生的错误的出错号。出错号不同，表示错误性质不同：

0：表示无错；1～999：供用户选用；1000以上：True BASIC提供的出错号。

EXTEXT$函数的功能：返回最近发生的错误的出错信息。这种出错信息既可能是用户预先定义的（如果用户曾定义该错误的出错信息），也可能是True BASIC系统提供的。如果程序运行没有出错，EXTEXT$将返回一个空串。

EXLINE$函数的功能：返回最近发生的错误所在位置。位置的表达形式为：

行号 IN 函数名/子程序名/主程序名

这里“行号”指的是错误所在的程序物理行号，注释行和空行均计算在内。行号之后列出错误所在的函数名、子程序名或主程序名。如果在函数或子程序的多重调用过程中发生错误，那么，EXLINE$ 将按照调用的相反顺序（自里层向外层）列出出错所在的函数名(或子程序名)和主程序名。True BASIC 将主程序名规定为“Main”。

上述三个函数均可出现在程序的任何地方。

例 1 下面的程序定义了函数 badfunc 和子程序 badsub。由于 m 值输入不当可能导致程序的运行错误。当发生错误时，在错误处理模块里显示错误类型、出错信息和出错位置。

程序:

```
DEF badfunc(n)=1/n                 --------------------
SUB badsub(m)                      Ok.RUN
    LET x=badfunc(m)               ? 5
    PRINT x                        0.2
END SUB                            Ok.RUN
INPUT m                            ? 0
WHEN error in                      3001
    CALL badsub(m)                 Divisin by Zero
USE                                1 in badfunc in Main,3 in badsub in Main,  8 in Main
    PRINT extype
    PRINT extext$
    PRINT exlineS
END WHEN
END
```

在程序中，被保护模块里的主程序调用了子程序 badsub，而子程序又调用了函数 badfunc。badsub 和 badfunc 均属被保护对象，其中的错误都将在错误处理模块里处理。

1.2.2 EXIT HANDLER 语句和 CAUSE ERROR 语句

EXIT HANDLER 语句仅用于出错处理模块。它的功能是退出当前的出错处理模块，但仍保持出错状态，如果该处理模块是套在另一保护模块之内（被保护模块嵌套使用），程序即转到外层的处理模块继续排错；如果该处理模块已是最外层，则由 True BASIC 系统处理；程序中断，在命令窗口显示出错信息。

CAUSE ERROR 语句用于模拟一个错误的发生，并定义该错误的出错号和出错信息。

语句格式:

CAUSE ERROR expr, expr$

其中，expr 为出错号，expr$ 为出错信息。

用户可以利用 CAUSE ERROR 语句引发一个错误，同时定义它的出错号和出错信息，并使程序转入出错处理模块。如果没有处理模块，将由系统处理这个错误。

例 2 对于如下程序:

```
DO while more data
   READ a, b
   LET x=a/b
```

```
        PRINT "Answer1="; x
        LET y=sqr(x)
        PRINT "Answer2="; y
        PRINT
    LOOP
    DATA 1, 2, 1, 0, a, b, 1, -2
    END
```

为了使它在运行出错时能得到相应的处理，使用嵌套 WHEN/END WHEN 的方法加以保护。

程序:

```
    ! A error handler
    DO while more data
        WHEN error in
            WHEN error in
                READ a, b
            USE
                EXIT HANDLER
            END WHEN
            !
            WHEN error in
                LET x=a/b
                PRINT "Answer1="; x
            USE
                CAUSE ERROR 200,
                "Division by zero.ignore them !"
            END WHEN
            LET y=sqr(x)
            PRINT "Answer2="; y
        USE
            IF extype=200 then
                PRINT extype
                PRINT extext$
                PRINT exline$
            ELSE IF extype=8101 then
                PRINT extype
                PRINT extext$; "ignore them !"
                PRINT exline$
                READ aS, b$
            ELSE
                EXIT HANDLER
            END IF
```

```
------------------------
Ok.RUN
Answer1=.5
Answer2=.707107

200
Division by zero. ignore them !
13 in Main

8101
Data item isn't a number. ignore them !
8 in Main

Answer1=-.5
SQR of negative number.
```

```
    END WHEN
    PRINT
LOOP
DATA 1, 2, 1, 0, a, b, 1, -2
END
```

程序进入 DO/LOOP 循环后，数值变量 a，b 读入第一组数据 1，2，没有运行错误，因此程序得到正常结果；读入第二组数据，由于 b 值为零，程序转入出错处理：引发一个出错号为 200，出错信息为 "Division by zero . ignore them !"的错误，并使程序转向外层出错处理模块，显示出错号、出错信息和出错位置；读入第三组数据时，由于数据类型与变量类型不符导致错误发生，进行如下出错处理：执行 EXIT HANDLER 语句，迫使程序转向外层处理模块进行出错处理，报告错误情况，并用 READ a$，b$ 语句将这组无效数据读掉；读入第四组数据，首先算出第一个结果，但在计算 x 的平方根时，由于 $x<0$ 导致运行出错。在做出错处理时，因这个错误的出错号既不等于 200 又不等于 8101，所以，执行语句 EXIT HANDLER 跳出处理模块由 True BASIC 系统处理：程序中断，报告出错信息 "SQR of negative number."，同时将光标移至出错行。

附录 2　程序的设计与调试技术

程序优劣的衡量标准一般包括以下几个方面：

(1) 正确性　程序能准确地实现数学模型。

(2) 逻辑结构　程序的逻辑清晰，层次分明，易于修改和调试。

(3) 阅读与操作　比较容易阅读和使用。

(4) 容错性　程序能较好地处理某些错误。

(5) 可移植性　程序不加修改（或稍加修改）便能在不同的计算机上运行。

(6) 运行速度　程序能迅速地完成任务。

当然，在设计程序时要顾全上述所有标准是困难的，但程序越接近上述标准，程序的质量就会越高。

一般，程序设计要经历以下几个步骤：

数学模型→算法设计→框图设计→程序编制→程序调试→程序运行。

2.1　程序的设计

给定数学模型后，应当把问题分解成若干个相对独立的子问题。选择和确定算法时，应作统筹安排。在用高级语言实现算法之前，应把程序的大致流程画成框图，这对大型程序的设计尤为重要。作框图时，先按程序功能画出粗框，其中仅包括各功能块。然后，采用"逐步求精"的方法，把各功能块逐渐细化，最后得到详细的框图。为具体编制程序提供依据。

清晰的逻辑是程序正确的前提，而有条理地书写程序则有助于表达清晰的逻辑。优良的程序格式不仅使程序易于阅读，不易出错，而且会给程序调试带来方便。书写程序时应注意：

(1) 采用"层次"(右缩)格式写程序；

(2) 语句定义符用大写字母，其他用小写字母；

(3) 变量或模块的命名要和它们所起的作用或功能相联系；

(4) 在关键语句和模块前(或后)加上适当的注释语句，说明它们的作用；

程序设计倡导简明精炼，不必追求巧妙，否则既费解又不易调试。要编好程序并不是一件容易的事，只有通过大量的实践，不断积累经验，才能逐步掌握编程的技术。在程序设计过程中注意下列几点，将有助于提高程序的质量。

(1)尽量避免与值为实数的表达式进行“相等”的比较。由于机器在进行实数运算时会产生误差，这种误差可能导致一个理论上是成立的关系变为不成立，从而引起条件判断失灵。

(2)常用的同值表达式可以预先计算，并将其值赋给一个变量，在以后的程序中，用这个变量代替原先的表达式。这不仅可以缩短程序，还可加快运行速度。

(3) 应将循环内部所有能够在循环外确定和完成的运算处理提到循环外进行，避免重复执行这些语句。

(4) 使用恒值变量。当程序中经常用到某个常数，可以预先把这个常数赋给一个变量(恒值变量)，凡出现该常数的地方便代之以相应的恒值变量。使用恒值变量便于对程序进行修改并确保该常数在程序中的一致性。

(5) 显示提示信息。以便用户随时了解程序进程和下一步的操作。

(6) 常用的运算处理可以编成函数或子程序。这样不仅能简化程序，且能使程序的逻辑结构清晰。使用外部子程序和函数，可以避免变量混淆；共用的子程序或函数应建在库中。

(7) 对可能发生错误的地方，预先加上出错处理。可以屏蔽错误，预防程序中断。

(8) 对于某个条件的判断，应该考虑到它所有可能的情况。

(9) 使用动态数组，便于根据需要调整内存用量。

2.2 程序的调试

程序写成后是否正确，有待于检查和验证。通常，一个较大的程序写完后，难免存在这样那样的错误。纠正这些错误的过程，就是程序的调试过程。

一个程序要运行，首先应该没有语法错误。调试程序的第一步就是找出并纠正所有的语法错误。只要将程序输入计算机运行，True BASIC 系统便首先检查程序是否存在语法错误。若程序有错，光标将停在第一个错误的位置上，并显示出错信息。用户可以根据这些信息了解错误的原因，改正错误。改正一个错误后，再次运行，便可找出其他的语法错误。

一个程序能运行，只能说明它没有语法错误，并不能保证它没有逻辑错误（注意，因为调试是一种抽样检验法，对程序进行调试往往只能证明它有错误，而不能证明它没有错误）。程序调试的难点是如何寻找其逻辑错误。

2.2.1 常见错误

导致程序逻辑错误的原因很多，常见的有：

(1) *算法错误*　例如数学模型的设计存在问题，或者模型的求解过程中有错误。

(2) *变量名(或函数、子程序名)*错误　分两种情况：一种是本应不同的两个变量，由于命名不慎，使变量重名，导致“两个变量”(事实上它们是同一变量)同值；另一种正好相反，本应是同一变量，由于误写，变成了两个不同的变量，写错的变量初值为零(字符串变量初值为“空”)。两种错误都可

能造成表达式计算错误，条件判断失灵，或死循环．后一种错误还经常表现为表达式的值为零．或因分母为零导致“溢出”．

(3) **变量没赋值** 由于变量没有赋值，其隐含值为零（或空），可能导致其他错误．

(4) **条件判断错误** 可分三种情形：第一种是条件错；第二种是列举的情形不完全；第三种是条件本身无错，但其中变量值有误而引起条件判断失灵．这三种错误都会导致判断失效或造成死循环．

(5) **下标越界** 数组下标的表达式写错或表达式中变量值有误都可能引起下标越界．

(6) **表达式错误** 表达式写错将引起表达式计算错误．最常见的是少加（或多加）括号，从而改变了原来的运算顺序，得出错误的结果，另一种情况是表达式中变量值错．

(7) **数据量和数据类型错误** 数据文件或语句的数据不够或数据量类型与欲赋值的变量类型不匹配引起的错误．

2.2.2 常用调试技术

调试是寻找和纠正程序错误的过程．常用的调试技术有：追踪，中断，强制和分块调试．下面分别讨论之．

(1) **追踪技术** 跟踪、检查程序是否按预定流程运行．可以在程序运行的“必由之路”设置一些输出语句，打印某种标志（类似于“路标”）．如果程序运行后，某个标志没有打印出来，就说明在对应的输出语句附近存在着逻辑错误．使用追踪技术可以缩小纠错的范围，多用于检验条件判断是否正确．顺便指出，对条件的每个分支都要进行检验，以免在某个分支隐藏错误．

(2) **中断技术** 在程序中设置断点，用以考察程序中某些变量的动态．断点是用 BREAK 命令（或 STOP 语句）设定，在调试过程中通常与 CONTINUE 命令配合使用．程序运行到断点时将暂停，这时用户可在命令窗口用 PRINT 语句将有关变量的当前值显示在屏幕上，中断技术常用于检查循环里某些变量的变化．

(3) **强制技术** 将某些变量强行置成常数，观察程序某一部分的运行状态和结果与预定情况是否相符，以便寻找错误源．这种技术多用于寻找一个程序模块的错误．

(4) **分块调试** 将程序分成若干个段(或模块)来调试．先对各段(或模块)单独进行调试，而后再逐次组合起来调试．这种调试技术对大型程序特别有用．

应当指出，利用打印语句 PRINT 打印标志或变量的中间值是程序调试的常用手段．由于它们能反映出程序的进行状态，所以被广泛应用于各种调试技术之中．

2.2.3 True BASIC 的调试方法

(1)设断点 中断技术在 True BASIC 里的具体实现是设置断点．作法如下：当需要在某行设置断点时，可在编辑窗口先用[F4]键在该行上做标记，然后在命令窗口键入 BREAK 命令．这时，在标记行的行头就出现“>”号，表示断点设在此行；若想取消断点只须再键一次 BREAK 命令．也可以在 BREAK 命令之后加一行号 n:

```
BREAK n
```

直接为程序的第 n 行加断点标志“>”．程序中断后，可在命令窗口用 CONTINUE 命令使程序自断点起恢复运行．

(2) **利用调试工具** True BASIC 版本 2.03 为了使用户更方便地调试程序，专门提供了一套调试软件：

```
TRACE.TRU      TRACE2.TRC      TRACE3.TRC
```

其中，TRACE2.TRC 和 TRACE3.TRC 是库文件．

调试命令的一般格式如下：

```
DO TRACE,option(var1, var2, ..., varN)
```

其中，Var1 为欲跟踪的变量名，一次允许同时跟踪多达 8 个变量，但只允许是简单变量；命令中的 option 可以是“step”，“slow”，“fast”，“break”四种选择之一．

step 表示单步跟踪，按一次空格键执行一个语句；slow 表示慢速跟踪；fast 表示快速跟踪；break 表示开始时快速跟踪，当满足给定条件时转为单步跟踪．

其实，选择只是影响跟踪速度，而跟踪方式都是相同的．option 不论是选择 step，slow 或 fast，在跟踪时利用按键都可以随意改变跟踪速度：按空格键，变为单步跟踪；按“s”键，变为慢速跟踪；按“f”键，变为快速跟踪．

在执行跟踪时，首先在屏幕的右边出现两个窗口，被跟踪的变量名及其内容显示在右上窗口，程序的运行结果(图形除外)显示在右下窗口，而屏幕的左边则显示被跟踪的程序段．若无特殊指明，整个程序均在跟踪的范围之内．跟踪开始时，有一条光带罩在第一条将要执行的语句上，此后，随着程序的跟踪执行，光带总是罩在下一条将要执行的语句上．因此，通过光带就可以了解程序目前运行到何处，而通过右边的两个窗口可以看到被跟踪的诸变量随程序的运行发生的变化以及程序产生的结果．

附录 3　True BASIC 的运行环境

3.1　硬件环境

True BASIC 版本 2.03 可以在配有 192K 以上内存的 IBM-PC 及兼容机上运行．它支持 CGA，EGA 等显示器，利用工作方式设置语句，可以设置 CGA，EGA 和 MONO 在内的多种图形和正文工作方式．当装有 8087 或 80287 数学协处理器时，True BASIC 能自动适应，即在进行算术运算时，能自动用上它们．但是，有没有数学协处理器对数值量而言，其内部表示是不同的．有数学协处理器时，数值量的内部浮点表示符合 IEEE 格式．

3.2　软件环境

3.2.1　西文系统

True BASIC 版本 2.03 可以运行在 PC-DOS(或 MS-DOS)2.00 版本以上的系统．

3.2.2　中文系统

True BASIC 不仅可以在西文操作系统中运行，还可以在某些中文操作系统如 CCDOS 2.10 和 CCDOS 2.13 等系统中运行．

具体操作步骤如下：

(1) 进入汉字操作系统；

(2) 在操作系统的提示符(设为 C>)下，调入 True BASIC 解释系统：

```
C>HELLO
```

(3) 在 True BASIC 的命令窗口的 Ok.提示符下，键入

```
MODE HIRES
```

汉字系统下的True BASIC屏幕显示效果与西文略有不同:

①西文状态下光标是闪烁的，而中文状态下光标是静止的;

②西文状态下命令和编辑窗口的分界线是由“–”构成的横线，而中文状态下，分界符是由汉字“哪”字构成的长串;

③在西文状态下，编辑窗口每行行头均有行标志，而中文状态没有行标志;

④在西文状态下，编辑窗口和命令窗口在屏幕上同时出现，而在11行的中文状态下，屏幕只能出现两个窗口之一，窗口的切换仍用F1和F2键；如果使用25行以上的汉字系统，则屏幕出现两个窗口与西文状态相同。

中文状态的True BASIC用法与西文状态基本相同，只是汉字仅能出现在字符串常量的位置上，不能作为关键字或变量名使用。另外，在某些工作方式下，屏幕绘图语句失效；在CGA的GRAPHICS和HIRES方式下绘图语句有效。

编辑时，如果不需要输入汉字，可在命令窗口键入MODE 80命令，将屏幕临时改为西文状态，便于编辑程序。当需要输入汉字字符时再利用MODE HIRES命令将屏幕还原成中文状态。另外，MODE HIRES和MODE 80用作语句时，应写为:

SET MODE “HIRES”和SET MODE “80”

例1 在CCDOS 2.10汉字系统支持下，在屏幕上画一个圆。程序运行时用汉字提示输入圆半径的值，画出圆后在圆心处显示汉字“圆心”。

程序:

```
CLEAR
INPUT PROMPT"请输入圆的半径值": R
SET MODE "GRAPHICS"
SET WINDOW -1.5, 1.5, -1, 1
SET COLOR "GREEN"
BOX CIRCLE -R, R, -R, R
SET COLOR "RED"
PLOT TEXT,AT 0,0: "圆心"
END
```

3.2.3 True BASIC的编译和汇集

通常，调入HELLO.EXE后，True BASIC即工作在解释状态，源程序调进内存。使用RUN命令，是在解释态下运行程序。由于True BASIC解释系统需要占据较大的空间，用户可用的空间就相对减少了。为了提高内存效益和加快速度，True BASIC提供了一种汇集手段。即：利用下面两个汇集程序(或包括EXEC.TRC)，就可将若干个经过True BASIC解释系统编译了的文件，汇集成一个可执行文件（扩展名为EXE)：BIND.EXE　BIND.RTP

下面举例说明汇集步骤。

设有一个求1～10阶乘值的程序（文件名为DEMO.TRU）和一个阶乘函数库程序(文件名为FUNCTION.TRU)。

文件DEMO.TRU:

```
LIBRARY "FUNCTION.TRC"
DECLARE DEF FACT
FOR i=1 to 10
```

文件FUNCTION.TRU:

```
EXTERNAL
DEF FACT(n)
    LET s=1
    PRINT FACT(i),
  NEXT i
  END
```

```
FOR t=1 to n
    LET s=s*t
NEXT t
LET FACT=s
END DEF
```

汇集：

(1)在操作系统状态下调入 True BASIC 解释系统；

(2)在 True BASIC 状态下，用 OLD 命令调入文件 DEMO.TRU；

(3)用 COMPILE 命令编译，并用 SAVE 命令存盘(文件名为 DEMO.TRC)；

(4)用 OLD 命令调入文件 FUNCTION.TRU；

(5)用 COMPILE 命令编译，并用 SAVE 命令存盘(文件名为 FUNCTION.TRC)；

(6)退出 True BASIC 系统，回到操作系统；

(7)在操作系统状态下，运行 BIND(设汇集程序存在 C 盘主目录)，即键入

```
C>BIND DEMO
```

在磁盘上形成一个已汇集的可执行文件 DEMO.EXE．这个文件可直接在操作系统下运行．

在汇集程序的过程中，已将所有用户程序用到的子程序和函数(包括库程序)都包含在汇集后的程序里．因此，执行汇集后的程序时就不再需要外部库文件了．汇集后的程序一般比 True BASIC 解释系统下占用的内存少，因而可以获得更大的存贮空间．但是，在需要汇集的程序里不能使用 CHAIN 语句．若要使用此功能可以采用另外的方式代替．将 True BASIC 提供的一个库程序 EXEC.TRC，加在程序开头

```
LIBRARY "EXEC.TRC"
```

调用其中的子程序

```
CALL EXEC(NAME$, ARG$)
```

就能实现终止当前程序，而装入另一个可执行的程序文件(EXE 文件或 COM 文件)．其中，NAME$ 是欲装入的文件的完整文件名表达式，而 ARG$ 是传递给欲装入程序的字符串参数(<128 个字符)，被装入的程序若要使用这个参数，应在程序开头加上 PROGRAM NAME(STRING$)语句，这样 ARG$ 的值就能传给 STRING$．这里 NAME$ 是一个任意的变量名，STRING$ 是接收参数的字符串变量名；如果不需要传递参数，则 PROGRAM 语句可以省略，调用时传入的参数(ARG$ 的内容)可以是一个空串．

此外，True BASIC 版本 2.03 还提供了另一个库程序 SHELL.TRU．它可使用户在 True BASIC 源程序这一级使用简单的语句就能够调用操作系统命令．使用它时只要在程序的开头加上语句

```
LIBRARY "SHELL.TRU"
```

即可调用其中的子程序 DOS(expr$)．这里 expr$ 是一个 DOS 命令的字符串表达式．例如，调用 DOS 显示磁盘目录的程序可以写成：

```
LIBRARY ″SHELL.TRU″
INPUT PROMPT ″Which driver ? ″: D$
LET D$ =D$[1:1]&″:″
CALL DOS(D$&″DIR / W″)
END
```

这个程序执行后，若用户从键盘上输入的驱动器符为“A”，则显示A盘的目录，若为“C”，则显示C盘目录.

但是，应该特别注意，要实现DOS的命令调用，必须事先将DOS的命令解释文件COMMAND.COM拷入当前目录中.

使用库程序SHELL.TRU，除了能用DOS(expr$)子程序调用DOS命令外，还可用子程序SHELL(expr$)调用DOS命令. 两者的差别在于：前者执行之前先清屏幕，而后者不清屏幕.

附录4 命令总结

4.1 系统命令

True BASIC系统命令包括编辑命令、文件处理命令和程序控制命令. 现分类归纳如下.

4.1.1 建立、保存和更新文件的命令

除已在第2章介绍的FILES，NEW，OLD，REPLACE，SAVE，UNSAVE命令外，还有一个磁盘文件换名命令：

RENAME 改变某一程序的文件名. 此命令有两种形式：

(1) RENAME newname 改变当前编辑窗口程序的文件名，不影响磁盘上任何文件.

(2) RENAME oldname，newname 将磁盘上的oldname文件改成名为newname的文件，不影响当前窗口的文件名。其中，"oldname"和"newname"还可包括一个磁盘设备名和目录名.

注意：第二种形式的RENAME命令只能重新命名一个文件到同一磁盘上.

4.1.2 编辑文件的命令

CHANGE 修改一个字或一短语. 例如，将内存中程序的所有“the”改为“this”，用命令

```
CHANGE the,this
```

但此命令并不改变某个字的一部分.

COPY 在文件中复制一部分到别的地方. 命令的最简单形式是：

```
COPY fromnum,tonum
```

fromnum表示一些行或行模块. True BASIC把fromnum所指的行模块通过COPY命令立即复制到tonum所指行的位置后面. 如命令：

```
COPY 150-200,400
```

将150～200行复制到400行后面. 还可以在fromnum的位置上使用函数、子程序或图画的名字. 如果已标记了行模块，则可用下面的简化形式：

```
COPY tonum
```

它将已标记的行模块复制到tonum所指的行位置之后.

DELETE 删除文件的一部分。用DELETE命令删除程序中已标记的行模块，可直接键入

```
DELETE
```

删除一个名为trace的程序段，可用

```
DELETE trace
```

DO FORMAT 按系统规定的格式理整当前的程序. True BSAIC系统盘上有一FORMAT程序，它可以按结构向右缩的标准格式重新整理一个True BASIC程序，并把关键字改为大写.

EDIT 对文件的一部分进行编辑，使编辑限制在一段程序、一个函数、一个子程序或图画程序中。例如：

EDIT func1

将编辑操作限制在名为 func1 的模块，也可以把编辑限定在用功能键 F4 标记的模块内。

INCLUDE　在当前文件中插入一个磁盘文件。所插入的文件总是放在光标所在行的后面。例如，想把一名叫“until1.tru”的文件加到当前程序中，可键入

INCLUDE until1

KEEP　保留文件的一部分，删除其余的部分。例如，删除当前文件中指定的行模块以外的程序行。键入

KEEP apple

这里 apple 是程序中名为 apple 的图形模块。执行上述命令后，仅保留 apple 图形模块。

LOCATE　寻找包含某个字或短语的所有程序行，并将它们显示在屏幕上。如果需要匹配字的某一部分，或区分字母的大小写，则被查寻的内容应用双引号引起来。当寻找有双引号引起来的内容时，可键入:LOCATE ″″′′。

MARK　给一个程序段加标记。当需要对某行模块(如函数、子程序、图画或某一程序段)进行编辑操作时，可采用 MARK 标记命令。标记后的模块，可以移动、复制或删除。例如，标记以 subn1 为名的子程序，可键入

MARK subn1

MOVE　移动文件的一部分到另一地方。MOVE 命令格式同 COPY 命令完全一样，区别仅在于 COPY 命令是复制，而 MOVE 命令是移动。

TRY　改变一个字或一短语。TRY 命令也用于全局修改。命令格式同 CHANGE 命令的格式一样。不同之处在于，它每改一次之前询问是否同意修改，Y (Yes) 表示改，N (No) 表示不改，Q (Quit) 表示不改且退出修改状态。

4.1.3　更改行号的命令

DO NUM　给当前程序加上行号。因为 NUM.TRU 程序在 True BASIC 系统盘上。运行 DO NUM 命令时一定要把装有 NUM.TRU 程序的盘放入驱动器中。如果当前程序已有行号，将重新安排行号。若程序小于等于 90 行，编号从 100 开始，行号增量为 10。程序多于 90 行，则起始行号从 1000 开始。

DO RENUM　将对已有行号的程序重新编号，并且能同时改变一般 BASIC 中的 GOTO 和 GOSUB 等语句的转向行号。其形式为:

DO RENUM first,step

first 是第一行的新行号，step 是行号递增量。如果不带后面的两个数值，则编号将从 100 开始，增量取值为 10。RENUM.TRU 程序在 True BASIC 系统盘上。

DO UNNUM　删除程序的行号及行号后面的空格。UNNUM.TRU 程序在 True BASIC 系统盘上。

4.1.4　调试程序的命令

BREAK(F8 键)　在程序中设置断点。命令形式:

BREAK　或　BREAK block

在程序的行标记上按 F4，然后在命令窗口键入 BREAK，标记行上出现“>”，即设置了一个断点。参数 block 是函数、子程序或图画名，说明断点在该行模块的首行，当程序执行到达断点时，光标就停在断点行上。如果断点设置在某一不可执行的程序行上（如注释行或 DIM 语句行），True BASIC 将把断点放在此行后的第一个可执行程序行上。如果想从程序行上去掉断点，在命令窗口再键入一次 BREAK 命令删去“>”断点标记。

CONTINUE　从断点处继续执行程序。程序运行因设有断点而暂停，可用此命令使程序继续运

行。命令形式：

```
CONTINUE
```

4.1.5 其他命令

BYE　退出 True BASIC 系统，返回磁盘操作系统。

COMPILE　编译当前内存中的程序，将源程序编译形成机器代码。编译后，True BASIC 系统将不保留源程序。如果打算保留源程序的副本，应该在编译前做好复制工作，编译后，仍可以使用 SAVE 和 RUN 等命令。SAVE 命令是把编译产生的代码记盘，隐含的文件扩展名为 TRC。用 OLD 命令可将它再调入内来运行，但不能阅读和修改。

DO　运行一个预处理程序，对当前程序进行某种处理。DO 命令，允许运行磁盘上的一种特殊的程序，这种程序用于检查或修改当前程序文件。由 DO 命令使用的特别程序与一般程序不同，它必须写成一个外部子程序(即以 EXTERNAL 开头的子程序，但无主程序)。这个外部子程序需要两个参数：一个字符串数组，后跟一个串变量。字符串数组装源程序，串数组的每个元素装一行被处理的源程序。后跟的串变量则保存来自 DO 命令的子程序名后的选择项的内容。如果子程序后没有选择项则该串变量的值为空。在用 DO 命令执行的程序中可以有多个外部过程，True BASIC 把程序中第一个外部过程作为“主过程”。

ECHO　使打印机或文件“响应”屏幕输出。命令细分为：

(1) ECHO　使打印机响应屏幕输出。键入此命令后，凡在命令窗口输出的信息，都原样传送到打印机输出。注意，在使用 ECHO 命令之前应先打开打印机。

(2) ECHO OFF　关闭响应。

(3) 命令后再加上“ >> ”　将执行命令的结果输出到打印机上。例如，将 FILES 命令的结果在打印机上打印出来，键入命令：

```
FILES  >>
```

则会在打印机上打印出以.TRU 和 .TRC 为扩展名的程序目录。

将程序 CIRCLE.TRU 的运行结果在打印机上打印出来，有两种方法：

第 1 种方法：

```
Ok. OLD  CIRCLE （将 CIRCLE.TRU 程序调入编辑窗口）
Ok. ECHO （使打印机响应屏幕输出）
Ok. RUN （运行程序，运行结果显示在屏幕上并且送往打印机）
Ok. ECHO  OFF （关闭打印机对屏幕输出的应响）
```

第 2 种方法：

```
Ok. OLD  CIRCLE
Ok. RUN  >>(程序运行结果在屏幕上显示出来，并且送往打印机 )
```

(4) ECHO　TO　filename　使文件 filename 响应屏幕的输出。键入此命令后，凡在命令窗口输出的信息被原样传送到指定文件中，加在其末尾。

如果仅需将程序的运行结果送往某一文件中保存，可用如下命令：

```
RUN  >>filename
```

将程序 ABC.TRU 的运行结果存到文件名为 AAA.TRU 的文件中，有两种方法：

第 1 种方法：

```
Ok. OLD  ABC.TRU （将程序 ABC 调入编辑窗口）
Ok. ECHO  to  AAA （使文件 AAA.TRU 响应屏幕输出）
Ok. RUN （运行程序，并将结果送往文件 AAA.TRU 中保存）
Ok. ECHO  OFF （关闭文件 AAA.TRU 对屏幕输出的响应）
```

第 2 种方法:

```
Ok. OLD ABC.TRU （将程序 ABC 调入编辑窗口）
Ok.ECHO to AAA(使文件 AAA.TRU 响应屏幕输出)
Ok. RUN >> （运行程序，并将结果送往文件 AAA.TRU 中保存)
```

ENTER 打开另一个子目录或打开指定目录名的子目录。命令形式:

ENTER directory 或 CD directory

例如: 打开 A 盘的 Sources 子目录，可键入:

ENTER a:\sources

HELP 问题解释。在命令窗口键入“HELP”，屏幕将暂时被清除，并显示 True BASIC 的简介。当按任意键时，将返回到清屏前的状态。另外; 键入“HELP TOPICS”时，将显示指定问题的清单。在特定问题下求助时，应在 HELP 后面给出问题的名称（大小写均可)。名称必须拼写正确，不能缩写，因为所有的解释均在 True BASIC 系统磁盘中，求助前必须装好系统磁盘。

INFO 在屏幕上显示下列信息项: 文件的行数和所占字节数; 文件的当前行; 当前磁盘所剩余的存储空间数（用字节表示）及所在的当前目录。例如，键入 INFO 命令，可得到类似下列的信息。

```
File size 39 lines,  719 bytes
At line 5
Free memory 489200 bytes
Directory c:\TRUEBAS
```

KEY 给键盘上的键定义新的含义。此命令只允许对特殊功能键作重新定义。对字母、数字和特殊符号键均不能再定义。例如，把 control-L 键定义成右移光标键，可采取以下具体步骤:

(1) 键入 KEY 并回车，True BASIC 等待重新定义;

(2) 由于打算将 control-L 定义成右移光标键，所以按 control-L 键;

(3) True BASIC 询问此键的新含义，此时按右移光标键 → ;

(4) 按 control-A 键: (凡重新定义某键的含义之后都必须用 control-A 作为结束)。

另外，可以把键的定义记录到文件中，也可以从文件中读出键的含义。KEY TO 把所有当前键的定义送入一个文件中。KEY FROM 则从某一文件中读出键的定义。具体形式为:

KEY FROM 文件名

KEY TO 文件名

LIST 在打印机上打印程序文件。用 LIST 命令可将文件以硬拷贝形式输出。既可以输出整个程序文件，也可以输出程序中的行模块。例如，

LIST 在打印机上输出整个文件。

LIST 100～999 在打印机上输出程序的第 100 行到第 999 行。

LIST sort 在打印机上输出行模块名为 sort 的子程序。

RUN 执行当前的程序。在运行一个程序以前，程序应在内存中。

SPLIT 在窗口分界线位置显示各功能键的功能提示。如果再使用一次 SPLIT 命令则提示消失。

! 键 使已标记行模块中的每行的前部都带上“!”号成为注释行。但是，如果行模块中含有注释行，则取消这些注释行前部的“!”号成为非“注释”行。具体操作:

(1) 按 F4 对行模块设置标记;

(2) 将光标移到已作标记的行模块的任一行的行标志符上，键入“!”键，则行模块的所有行都被打上“注释”标记“!”，再按 F4 消除行模块标记。

如果要恢复注释行成为程序行，可按 F4 对这段注释行设置标记，键入“!”键将消除该行模块中的所有“注释”标记“!”。

程序在调试过程中，有时需要跳过一段程序，这时可用“!”键，使之成为注释段。

TO　移动光标到某行。可把光标移到某行模块的开始位置。例如，

TO heapsort

把光标移到名为 heapsort 程序模块的第一行上，如果 TO 命令后面给出一个行号，那么，光标将移到该行；如果程序没有行号，那么，行号即为程序行的自然数顺序号。

此外，还有一种特殊情况，单行批命令。即允许在一行上键入若干命令（用分号将这些命令隔开），例如，

```
OLD flash; RUN; CHANGE 50, 100; RUN
```

当执行这些命令后将显示“Ok”提示符，如果其中某个命令有错，那么该命令及其剩余的部分将被抛弃。

4.2　作为命令使用的 True BASIC 语句

True BASIC 的有些语句也可作为命令来使用。一般情况下，单行操作性语句都可作为命令使用（但象 DIM，DEF 和 OPTION 等不能立即响应的语句不能作命令使用。另外，判断结构和循环结构也不能作为命令）。

下面是可以当作命令用的 True BASIC 语句：

```
ASK      BOX          CLEAR    GET    INPUT
LET      LINE INPUT   PAUSE    PLAY   PLOT
PRINT    SET          SOUND
```

仅当运行一个未编译过的程序之后，下列 True BASIC 语句作为命令才可使用数组、读（或恢复）数据、打开（或关闭）通道以及调用用户定义的子程序等。运行一程序后可作为命令的语句：

```
CALL     CLOSE        RESET    MAT
OPEN     READ         ERASE    WINDOW
WRITE    DRAW         RESTORE
```

这些命令对调试程序是很有用的。

附录5　语 法 补 充

下面列出本教程未介绍或仅在程序举例中引用的 True BASIC 语法内容。

5.1　简 单 语 句

ASK MARGIN var　将屏幕当前窗口边界宽度(每行字符个数)的值赋给 var。

ASK ZONEWIDTH var　将当前屏幕两标准位置之间的区域宽度值赋给 var。

LET var$[i:j]=expr$　用串表达式 expr$ 来代替 var$ 的第 i 个到第 j 个字符，若 i<1，则用 1 代替，若 j 超过串的长度则用其长度代替；若 j<i 则插入到第 i 个字符前的位置上，不删掉任何字符。

OPTION NOLET　执行该语句后，允许其后的赋值语句省略关键字 LET。

PAUSE expr　让程序暂停 expr 秒钟，expr 必须是数字。

SET MARGIN expr　设置屏幕当前窗口的边界宽度为 expr。

SET ZONEWIDTH expr　设置屏幕两标准位置之间的区域宽度。

注意：边界和区域宽度不能少于一个字符，边界宽度不得小于区域宽度。

STOP　暂停程序的运行．

5.2　矩阵语句

MAT LINE INPUT array1$, array2$, …

MAT LINE INPUT PROMPT expr$:array1$, array2$, …

从键盘上读入逻辑行(以回车符结束)并将它们赋给数组的各元素．数组必须是字符串型数组。PROMPT 选项中的提示串仅为头一个输入作显示．后继的输入以“?”提示．

MAT PRINT USING format$:array1, array2, …　按格式串指定的形式显示数组元素．

5.3　程序部件

CHAIN expr$

CHAIN expr$ WITH(arg$)

CHAIN expr$, RETURN

CHAIN expr$ WITH(arg$), RETURN

PROGRAM name

PROGRAM name(var$)

当执行 CHAIN 语句时，停止当前的程序，转去执行 CHAIN 语句指定的程序．字符串表达式 expr$ 给出要链接程序的文件名．

所有的通道(除屏幕／键盘通道外)在 CHAIN 语句执行时都被关闭，全部变量的存储单元都被释放．若使用“CHAIN expr$, RETURN”语句，被链接的程序执行完后，将返回到原来的程序（这时原来的程序状态被保存在存储器中）．使用带 RETURN 的链接语句并不节省存储空间。

被链接的程序必须用一个 PROGRAM 语句开始．CHAIN 语句中若带字符串 arg$，该字符串将转给被链接程序．此时必须在被链接程序的 PROGRAM 语句中列出相应的链接参数 var$。例如：

```
LET exzmple$ = Time$
CHAIN "chainee" with(example$)
END
PROGRAM chainee(string$)
PRINT string$
END
! 程序执行结果将显示 Time$ 的值
```

5.4　图

ASK COLOR var　将当前的前景颜色号赋给数字变量 var．

ASK COLOR var$　将当前的前景颜色名赋给字符串变量 var$．

ASK COLOR MIX(c), r, g, b　将 c 号颜色中红绿兰 3 个基色的强度分别赋给 r, g, b。该语句仅用于 EGA 显示器．

ASK CURSOR var$　将当前光标状态赋给字符串变量 var$．var$ 的值为“on”(打开)或“off”(关闭)．

ASK CURSOR x，y　将当前光标的行列位置分别赋给数字变量 x 和 y。

ASK BACK var（或 ASK BACKGROUND COLOR var）　将当前的背景颜色号赋给数字变量 var。

ASK BACK var$（或 ASK BACKGROUND COLOR var$）　将当前的背景色赋给字符串变量 var$。

ASK PIXELS x，y　将当前图形窗口水平和垂直方向的象数个数分别赋给 x，y。

GET MOUSE: x，y，s　如果计算机安装了鼠标器，可使用该语句读取图形窗口的坐标。其中 x，y 返回坐标值，s 返回鼠标器在该点的状态:0-没按按钮，1-拖走，2-按下按钮，3-释放按钮。

SET COLOR MIX(c)，r，g，b　定义第 c 号颜色。其中 r，g，b 为红绿兰 3 个基色的强度。强度范围从 0 到 1。该语句仅用于 EGA 显示器。

SET CURSOR expr$　打开或关闭屏幕光标。expr$ 可以是"on"(打开)或"off"(关闭)。

BOX SHOW Var$ AT x_{min}，y_{min} USING"and"(或"or"或"xor")　扩展的 BOX SHOW 语句，将 Var$ 的图形与屏幕指定位置上的背景图形叠加，叠加方式可指定为:"与"("and")或者"或"("or")或者"异或"("xor")。

5.5　文件处理

下面的语句中，#expr 表示一个在 1～999 之间的通道号。

ASK #expr: ACCESS var$　询问文件的存取方式。"INPUT"或"OUTPUT"或"OUTIN"将传递到字符串变量 var$ 中。

ASK #expr: FILESIZE var　询问文件的大小。文件的大小将赋给 var，正文文件和字节文件是用字节数表示大小，记录文件由记录的个数表示大小。

ASK #expr: MARGIN var　将正文文件一行的宽度赋值给 var。

ASK #expr: NAME var$　询问文件的文件名。字符串变量 var$ 被赋予文件名。

ASK #expr: ORGANIZATION var$　询问文件的组织形式 。"TEXT"或"RECORD"或"BYTE"被赋予 var$。

ASK #expr: POINTER var$　询问文件指针位置，Var$ 被赋予指针的位置:"BEGIN"或"MIDDLE"或 "END"。

ASK #expr: RECORD var　询问文件指针当前具体位置，其值赋给数字变量 var。对于正文文件和字节文件其值表示字节数。记录文件是记录数。

ASK #expr: RECSIZE var　询问记录文件记录的长度。var 被赋予记录的长度值。对于正文文件和字节文件其值为零。

ASK #expr: ZONEWIDTH var　将正文文件标准位置之间的区域宽度赋值给 var。

注意: MARGIN 和 ZONEWIDTH 仅适用于正文文件。

MAT INPUT #expr: array1，array2，…

MAT LINE INPUT #expr: var1$，var2$，…

MAT READ #expr: array1，array2，…

MAT PRINT #expr: array1，array2，…

MAT PRINT #expr，USING format$: array1，array2，…

十五[illegible]

PRINT #expr: USING format$:var1, var2, …

以上为用于文件处理的矩阵语句和自选格式输出语句。

某些 SET 语句可简写为 RESET 语句:

标准形式	缩写形式
SET #expr: POINTER BEGIN	RESET #expr: BEGIN
SET #expr: POINTER END	RESET #expr: END
SET #expr: POINTER NEXT	RESET #expr: NEXT
SET #expr: POINTER SAME	RESET #expr: SAME
SET #expr: RECORD expr	RESET #expr: RECORD expr

SET/ASK DIRECTORY var$ 将当前所处磁盘目录的目录名设为/赋给 var$。

SET/ASK NAME var$ 将当前编辑窗口所用的文件名设为/赋给 var$

此外，在 OPEN 语句中，ORGANIZATION 可缩写成 ORG。

5.6 LOCAL 语句和 OPTION TYPO 语句

LOCAL var1, var2, … 在内部子程序或内部函数中可用 LOCAL 语句对变量(形式参数和公用变量除外)进行说明。说明后的变量成为局部变量。当进入该函数或子程序时，这些变量被初始化为 0 或空串。LOCAL 语句可以说明数组，说明时需标出各维的上下界。

OPTION TYPO 如果在主程序及各程序模块中通过 LOCAL 语句对所有变量(形式参数和公用变量除外)进行了说明，则可在主程序前部使用 OPTION TYPO 语句对程序变量进行检查。程序运行时，凡是未出现在 LOCAL 语句中的变量均会被查找出来，并给出出错信息“Unknown variable”。因此，借助 LOCAL 和 OPTION TYPO 可查出编程中写错的变量名。

5.7 系统函数

5.7.1 数学函数

DIVIDE(x, y, q, r) 返回 x/y 的商 q 和余数 r。用 CALL DIVIDE(x, y, q, r)调动。

EPS(x) 返回计算机内表示 x 时的误差上限。

FP(x) 返回 x 的小数部分。

IP(x) 返回 x 的整数部分。当 x>0 时为 INT(x)，否则为-INT(-x)。

REMAINDER(x, y) 返回 x/y 的余数。

TRUNCATE(x, n) 返回 x 取 n 位小数点的近似值。

5.7.2 三角函数

ANGLE(x, y) 返回 X 轴与点(x, y)之间夹角的度数。

5.7.3 字符串函数

CPOS(a$, b$)和 CPOS(a$, b$, n) 返回 b$ 中的字符在 a$ 中(第 n 个字符以后)的出现位置。

CPOSR(a$, b$) CPOSR(a$, b$, n) 返回 b$ 中的字符在 a$ 中从右侧开始(第 n 个字符以后)的出现位置。

NUM(a$) 返回与 a$ 相应的数值。其中 a$ 应是由 NUM$ 函数建立的

NUM$(x)　返回按 IEEE 格式将 x 转换后所得的 8 字节字符串。

NCPOS(a$，b$)和 NCPOS(a$，b$，n)　返回 b$ 中的字符在 a$ 中(第 n 个字符以后)不出现的起始位置。

NCPOSR(a$，b$)和 NCPOSR(a$，b$，n)　返回 b$ 中的字符在 a$ 中从右侧开始(第 n 个字符以后)不出现的位置。

POSR(a$，b$)和 POSR(a$，b$，n)　返回 b$ 在 a$ 中从右侧开始(第 n 个字符以后)的出现位置。

5.7.4　日期和时间函数

DATE　以 YYMMDD 的数值形式返回计算机的系统日期。

DATE$　以“YYYYMMDD”的字符串形式返回计算机的系统日期。

TIME$　以“HH:MM:SS”的字符串形式返回计算机的系统时间。

附录6　出错信息

True BASIC 的错误信息分为下述 3 类:

(1) 命令错。即输入了不正确的命令。

(2) 编译时出错。即编译源程序时所出现的错误。

(3) 运行时出错。即程序开始运行时所捕捉到的错误。

下面是按字母顺序列出的出错信息表。

Argument types don't match　在子程序未定义之前调用了子程序，或调用时参数个数不对，或参数类型不一致。

Bad find items; try using quotes　寻找一个包含逗号或双引号的字符串时，未用双引号括住。

Badly formed using string　在 USING$ 函数或 PRINT USING 语句中格式串有错。

Badly formed input line　在回答 INPUT 语句时格式不对，如丢失了双引号。

Cant't change that key　不允许重新定义一般显示字符键的含义。

Can't continue　程序中断后，凡属下列情况，CONTINUE 命令无效:

(1) 继续运行之前修改了程序; (2) 程序是由于出错而停止运行的; (3) 使用 CONTINUE 命令之前用过 DO 命令。

Can't copy region itself　不允许复制自身的某些程序行。

Can't do graphics on this computer　所用机器没有图形功能，不能画图。

Can't edit compiled program　程序被编译后就不能再被编辑。

Can't erase file not opened as outin　不能将 ERASE 语句用于非 OUTIN 方式的文件。

Can't find library name　未找到文件库 name。

Can't help with that.try help topics　没有 HELP 文件或无法回答所求助的问题。

Can't input from output file　OUTPUT 方式的文件不能用来读取数据。

Can't invert singular matrix　不能对奇异矩阵使用 inv 函数求逆。

can't output to input file　INPUT 方式的文件不能用来写入数据。

Can't print to middle of text file　数据不能写到正文文件的中部。

Can't set recsize on non-empty record file	不能对非空记录文件设置记录长度.
Can't set window in picture	在图画中不能重新设置窗口。
Can't use angle(0, 0)	函数 ANGLE(a, b) 中，参数至少要有一个非零.
can't use this statement here	在这里不能使用该语句.
Can't use#0 here	#0 通道总是打开的，不能对其使用 OPEN 和 CLOSE 语句.
Channel is already open	已经打开的通道不能再打开.
Channel isn't a window	所用的通道是文件通道，不是窗口通道。
Channel isn't open	使用了没有打开的通道.
channel number must be 1 to 1000	通道号的数值必须在 1 到 1000 范围内.
Constant too large : constant	常数太大.
Cursor set out of bounds	光标位置（行，列），超出了窗口的范围.
Data item isn't a number	DATA 语句给出的数据项不是数值数据.
Data item isn't a string	DATA 语句给出的数据项不是字符串数据.
Det needs a square matrix	DET 函数仅用于方阵.
Disk full	磁盘满了，不能完成向磁盘写入文件的操作.
Diskette removed, or wrong diskette	磁盘已从驱动器中取出. 或者磁盘有错.
Division by zero	表达式中出现被零除的情况.
Ending does't match beginning	结构语句的开头与结尾不匹配.
Error in play string	在 PLAY 语句中乐谱字符串有错.
Expected “thing”	光标所指的位置上不是 True BASIC 所要求的符号。
Expected relational operator	光标所指位置上，必须放上某关系运算符.
File already exists	已经存在的文件，不能再建立.
File is read or write protected	文件处于读写保护状态，不能对其进行读或写.
File pointer out of bounds	改变文件指针时，指针比 1 小或者超出了文件结束符.
File's record size does't match open recsize	设置的记录长度与实际尺寸不匹配.
IDN must make a square matrix	单位矩阵必须是方阵.
Illegal array bounds for name in routine	在数组定义中，上界不得小于下界.
Illegal channel	通道使用不合法
Illegal data	DATA 语句书写不正确. 语句中两个数据之间必须用逗号隔开.
Illegal exit	退出循环的语句与当前循环的类型不吻合 .
Illegal expression	光标所指的表达式有不符合规则的地方.
Illegal keyword	光标所指的语句关键字出错.
Illegal line number	行号书写不合法. 如在一个带行号的程序中，有一个无行号行等.
Illegal number	所写的数据不符合 True BASIC 语法.
Illegal option	在使用选择语句（OPTION）时可能拼写出错.
Illegal parameter	子程序或函数中的参数出错.
Illegal statment	光标所指处的语句非法.
Improper ORD string	使用 ORD 函数时，所给的自变量不是单字符而是多字符.
Improper NUM string	NUM 函数的自变量（字符串）不能表示为 IEEE 8 字节的浮点数.

Input item bigger than recsize　使用文件时某些记录长度比文件定义的记录长度要大.

INV needs a squre matrix　求逆矩阵只限于方阵

Lbound out of range　数组 A 为 n 维，当 m>n 时，使用 LBOUND(A，m)则出错。.

LOG of number<=0　对数函数的自变量不能取负数。

Margin less than zonewidth　SET MARGIN 语句设置的 margin 比 zonewidth 小。

Mismatched array sizes　数组尺寸不匹配.

Missing end statment　程序缺少 END 语句.

MOD and REMAINDER can't have 0 as 2nd argument　求模 MOD 和求余数 REMAINDER 函数不允许用零作为它们的第二个参数.

Must be a function name　使用 DEF 或 FUNCTION 语句时，没给函数名.

Must be a number　每个 CASE 后面的测试项必须是数字值常数.

Must be a picture name　DRAW 语句必须用图画名，不能用其他的名字.

Must be a string contant　该 CASE 后的测试项必须是字符串常数.

Must be a subroutine name　CALL 必须后随子程序名.

Must be a variable　在应当使用变量的地方，却使用了表达式或子程序名.

Must be an array　应该使用数组而不是简单变量.

Must be byte file for read bytes　不能对正文文件或记录文件使用 READ BYTES 语句.

Must be byte file or empty file for set recsize　SET RECSIZE 仅能用于字节文件或空文件.

Must be record or byte file for set record　SET RECORD 不能用于正文文件.

Must be record or byte file　必须是一个记录或字节文件。

Must be text file　必须是一个正文文件.

Must set recsize before write　在向一空记录文件写入信号之前要定义其长度.

Name can't be redefined　名字不能重复定义。

Negation number to non-integral power　计算 n 时，若 n 为负数，则 x 必须是整数.

No case selected，but no case else　对每个 CASE 测试都不成功时，应有 CASE　ELSE 语句块.

No get mouse on this computer　所用计算机无鼠标定位器，使用 GET MOUSE 语句出错.

No main program　程序中仅有函数，图画或子程序，没有主程序.

No saved copy-still want to compile?　当运行 COMPILE 命令时，提醒做备份.

No such color　设置的颜色名不符合规定.

No such directory　键入的子目录磁盘中不存在.

No such file　文件名不对或磁盘上没有该文件.

No such function on subroutine　函数或子程序不存在.

No such line numbers　命令中所给出的行号与实际中行号不符.

No using item for output　输出时没有格式描述串.

On index out of range，no else given　使用 ON GOTO 或 GOSUB 语句时，分支号超界.

Out of memory　需要的内存量太大，超出了机器的内存范围.

Output item bigger than recsize　向记录文件写数据时，不能超过记录的长度.

Overflow　超出了计算机所能处理的数值范围（溢出）.

Please say what lines you mean　MOVE 命令时，没有说明所要移动的行位置。

Please try “CHANG old, new”　修改短语时，同时给出原来的短语和要改的短语。

Please try “DO filename”　使用 DO 命令时，应给出文件名。

Please try“INCLUDE filename”　使用 INCLUDE 命令时，应给出文件名。

Please try “OLD filename”　使用 OLD 命令时，必须给出所要调入的文件名。

Please try “SAVE filename” or “REPLACE filename”　将当前文件存盘时，必须给出文件名。

Please try “ENTER filename”　执行 ENTER 命令时，须给出子目录名。

Please try RENAME new “ or ”RENAME old, new”　执行 RENAME 命令时，须给出要改名的文件名。

Please try ”SCRIPT filename, arg, arg…”　执行 SCRIPT 命令时，未给出文件名或所给参数不正确。

Please try to “ECHO” or “ECHO TO FILENAME” or “ECGI OFF”　键入的 ECHO 命令不符合其三种命令形式之一。

Please type line numbers as 100 or 100～150　DELETE 命名后的参数应是一个行号或由行号标示的行模块。

Program stopped　程序被迫停止运行。

Program unit too large　程序中某程序单元已超出了 64k 中间代码限制。

Repeat$ count<0　使用 REPEAT$(a$, n) 函数时，n 不能小于零。

Reading past end of data　执行 READ 语句时，读数超界。

Reading past end of file　要从文件中读数据时，读数超过文件尾。

Return without gosub　执行 RETURN 语句时，没有对应的 GOSUB 语句。

Screen bounds must be 0 to 1　设置的屏幕边界必须在 0 到 1 的范围之内。

Screen minmum >= maxmum　设置窗口边界时，右边界须大于左边界，上边界须大于下边界。

Size index out of range　数组 A 为 n 维，当 m>n 时，调用函数 SIZE(a, m) 出错。

SQR of negative number　不能对负数求平方根。

Statment outside of program　程序 END 语句后，只能是外部子程序、函数或图画。

String given instead of humbe　需要输入数值时，所给的却是字符串，类型错。

String too long　所建立的串长于 32000 个字符。

Subscript out of bounds　数组的下标超出了所定义的界限。

TAB column not between 1 and margin　使用 TAB 函数时，参数不在允许范围之内。

The compile command is just “compile”　使用 COMPILE 命令时，不能附带其他内容。

The continue command is just “continue”　使用 CONTINUE 命令时，不能附带其他内容。

This must first appear in a DIM or DECLARE DEF　数组和外部函数都必须先定义后使用。

Too few input items　输入的项数比 INPUT 语句中要求的项数少。

Too many shared channels in XXX　共用通道太多，只能在 1000 个以内。

Too many channels open　打开的通道太多(最多只能打开 10 个通道)。

Too many input items　输入的项数比 INPUT 语句中要求的项数多。

Trouble using disk or printer 磁盘或打印机出了问题。

Type is wrong for name in routine 调用子程序时，子程序名写错或者在调用时，参数类型出错。

Ubound out of range 数组 A 为 n 维，当 m>n 时，调用 UBOUND(A，m) 时出错。

Undefined public variable XXX insub 子程序中使用了未定义的变量。

Undefined routine name in routine 不能在子程序中再定义子程序、函数或图画。

Unknown open option 在 OPEN#expr: 右边的选择部分没有给出或拼写错误。

Unknown variable 变量未在 LOCAL 语句中列出。

Val string isn't a proper number VAL 函数中，所给定的串必须表示一适当的数。

What? (please type help) 键入的命令不能识别。

Window minimum = maxmum 所设置的垂直或水平方向的最大值不应与最小值相等。

Wrong number of arguments 调用子程序、函数或图画时，参数数值有错。

Wrong number of dimensions 数组的维数与 DIM 语句定义的数组维数不符。

Wrong type 所定义的量必须与所使用的量在类型上一致。

Wrong type of file 文件类型错误。

You have two public verlables named XXX 不能有重名的公共变量。

You have two routines called name in routine 子程序重名定义是非法的。

Zero to negative power 计算 0 的 n 次幂时，n 不能小于零。

Zonewidth out of range 所设置的 zonewidth 比 1 小或比 margin 大（越界）。

附录 7-1 ASCII 码

Decimal Value	Charactes	Decimal Value	Charactes	Decimal Value	Charactes
0	(null)	48	0	96	`
1	☺	49	1	97	a
2	☻	50	2	98	b
3	♥	51	3	99	c
4	♦	52	4	100	d
5	♣	53	5	101	e
6	♠	54	6	102	f
7	•	55	7	103	g
8	◘	56	8	104	h
9	○	57	9	105	i
10	◙	58	:	106	j
11	♂	59	;	107	k
12	♀	60	<	108	l
13	♪	61	=	109	m
14	♫	62	>	110	n
15	☼	63	?	111	o
16	►	64	@	112	p
17	◄	65	A	113	q
18	↕	66	B	114	r
19	‼	67	C	115	s
20	¶	68	D	116	t
21	§	69	E	117	u
22	▬	70	F	118	v
23	↨	71	G	119	w
24	↑	72	H	120	x
25	↓	73	I	121	y
26	→	74	J	122	z
27	←	75	K	123	{
28	∟	76	L	124	|
29	↔	77	M	125	}
30	▲	78	N	126	~
31	▼	79	O	127	⌂
32	space	80	P	128	Ç
33	!	81	Q	129	ü
34	"	82	R	130	é
35	#	83	S	131	â
36	$	84	T	132	ä
37	%	85	U	133	à
38	&	86	V	134	å
39	'	87	W	135	ç
40	(	88	X	136	ê
41	)	89	Y	137	ë
42	*	90	Z	138	è
43	+	91	[	139	ï
44	,	92	\	140	î
45	-	93	]	141	ì
46	.	94	^	142	Ä
47	/	95	_	143	Å

续表

Decimal Value	Charactes	Decimal Value	Charactes	Decimal Value	Charactes
144	É	192	└	240	≡
145	æ	193	┴	241	±
146	Æ	194	┬	242	≥
147	ô	195	├	243	≤
148	ö	196	─	244	⌠
149	ò	197	┼	245	⌡
150	û	198	╞	246	÷
151	ù	199	╟	247	≈
152	ÿ	200	╚	248	°
153	Ö	201	╔	249	∙
154	Ü	202	╩	250	·
155	¢	203	╦	251	√
156	£	204	╠	252	ⁿ
157	¥	205	═	253	²
158	₧	206	╬	254	■
159	ƒ	207	╧	255	(blank)
160	á	208	╨		
161	í	209	╤		
162	ó	210	╥		
163	ú	211	╙		
164	ñ	212	╘		
165	Ñ	213	╒		
166	ª	214	╓		
167	º	215	╫		
168	¿	216	╪		
169	⌐	217	┘		
170	¬	218	┌		
171	½	219	█		
172	¼	220	▄		
173	¡	221	▌		
174	«	222	▐		
175	»	223	▀		
176	░	224	α		
177	▒	225	β		
178	▓	226	Γ		
179	│	227	π		
180	┤	228	Σ		
181	╡	229	σ		
182	╢	230	µ		
183	╖	231	τ		
184	╕	232	Φ		
185	╣	233	Θ		
186	║	234	Ω		
187	╗	235	δ		
188	╝	236	∞		
189	╜	237	φ		
190	╛	238	ε		
191	┐	239	∩		

* 字符 128～255 是 IBM-PC 机上扩展的字符。

附录 7-2 用 GET KEY 语句获得的键码

	F1	F2	ESC	! 1	@ 2	# 3	$ 4	% 5	^ 6	& 7	* 8	(9	) 0	_ -	+ =	←	Num Lock	Scroll Lock
Normal	315	316	27	49	50	51	52	53	54	55	56	57	48	45	61	8	–	–
Shift	340	341	27	33	64	35	36	37	94	38	42	40	41	95	43	8	–	–
Control	350	351	27	–	259	–	–	–	30	–	–	–	–	31	–	127	–	STOF
Alternate	360	361	–	376	377	378	379	380	381	382	383	384	385	386	387	–	–	–

	F3	F4	⇤ ⇥	Q	W	E	R	T	Y	U	I	O	P	{ [	}]	7 Home	8 ↑	9 PgUp	–
Normal	317	318	9	113	119	101	114	116	121	117	105	111	112	91	93	327	328	329	45
Shift	342	343	271	81	87	69	82	84	89	85	73	79	80	123	125	55	56	57	45
Control	352	353	–	17	23	5	18	20	25	21	9	15	–	27	29	375	–	388	–
Alternate	362	363	–	255†	273	274	256†	276	277	278	279	280	281	–	–	§	§	§	–

	F5	F6	Ctrl	A	S	D	F	G	H	J	K	L	: ;	" '	~ \	↵	4 ←	5	6 →
Normal	319	320	–	97	115	100	102	103	104	106	107	100	59	39	96	13	331	–	333
Shift	344	345	–	65	83	68	70	71	72	74	75	76	58	34	126	13	52	53	54
Control	354	355	–	1	•	4	6	7	8	10	11	12	–	–	–	10	371	–	372
Alternate	364	365	–	286	287	288	289	290	291	292	293	294	–	–	–	–	§	§	§

	F7	F8	⇧	| \	Z	X	C	V	B	N	M	<	>	? /	⇧	PrtSc *	1 End	2 ↓	3 PgDn
Normal	321	322	–	92	122	120	99	118	98	110	109	44	46	47	–	42	335	336	337
Shift	346	347	–	124	90	88	67	86	66	78	77	60	62	63	–	‡	49	50	51
Control	356	357	–	28	26	24	3	22	2	14	13	–	–	–	–	–	373	–	374
Alternate	366	367	–	–	300	301	302	303	304	305	306	–	–	–	–	–	§	§	§

	F9	F10	Alt		Caps Lock	0 Ins	Del	+
Normal	323	324	–	32	–	338	339	43
Shift	348	349	–	32	–	48	46	43
Control	358	359	–	32	–	–	–	–
Alternate	368	369	–	32	–	§	–	–

Key

Normal
Shift
Control
Alternate

–	No character
•	Suspend output
†	Unreliable (255, 256)
‡	Print Screen
§	Alt and the numberic keypad generate any character 0–255.

参 考 文 献

1 Jhon G.Kemeny，Thomas E.Kurtz. True BASIC. 1985 年

2 Jhon G.Kemeny，Thomas E.Kurtz. True BASIC 2.03 版增补说明. 1988 年

3 陈松乔. 程序设计技术基础. 长沙：中南工业大学出版社，1988 年

4 华东师范大学计算机科学系，郑州工学院科研所，BASIC程序设计和算法基础. 上海：上海科学技术文献出版社，1982 年

5 谭浩强，田淑清. FORTRAN−77结构化程序设计. 北京：高等教育出版社，1985年

6 王诚，苏云清，赵毓升. PASCAL程序设计及其应用. 北京：清华大学出版社，1983年

7 仲萃豪，冯玉琳，陈友君. 程序设计方法学. 北京：北京科学技术文献出版社，1985年

8 N. 沃思〔瑞士〕著；曹德和，刘春年译. 算法+数据结构=程序. 北京：科学出版社，1984 年

9 D. E. 克鲁特〔美〕著；管纪文，苏运霖译. 计算机程序设计技巧（第一卷），基本算法. 北京：国防工业出版社，1980 年

Document generated by Anna’s Archive around 2023-2024 as part of the DuXiu collection (https://annas-blog.org/duxiu-exclusive.html).

Images have been losslessly embedded. Information about the original file can be found in PDF attachments. Some stats (more in the PDF attachments):

```
{
  "filename": "MTA5NDM0ODUuemlw",
  "filename_decoded": "10943485.zip",
  "filesize": 17478784,
  "md5": "14089346a0eff7d3674fd73fb00c8f5d",
  "header_md5": "9201a572dda35c93810dc828bdfc8709",
  "sha1": "08eac639333c444c118c7946e1de47e6d16d7bb3",
  "sha256": "082f510e364d79506e2c5c0fca395c83a11fd9ead010f8d74f5f9be18041110e",
  "crc32": 4200931215,
  "zip_password": "",
  "uncompressed_size": 18401477,
  "pdg_dir_name": "10943485_TrueBASIC\u2502\u2560\u2568\u2265\u2554\u03a6\u255d\u255e\u255c\u2560\u2502\u2560",
  "pdg_main_pages_found": 228,
  "pdg_main_pages_max": 228,
  "total_pages": 235,
  "total_pixels": 1647775680,
  "pdf_generation_missing_pages": false
}
```